Selection & Projection

Other Relational Algebra operations

Selection

Restriction

Projection

Union

Union All

Intersect

Minus

Join

Select Operation

- The select operation selects tuples that satisfy a given predicate.
- It is denoted by sigma (σ).
- Notation: σ p(r)

Where

σ is used for selection prediction

r is used for relation

p is used as a propositional logic formula which may use connectors like: AND OR and NOT. These relational can use as relational operators like =, ≠, ≥, <, >, ≤.

BRANCH_NAME	LOAN_NO	AMOUNT
Downtown	L-17	1000
Redwood	L-23	2000
Perryride	L-15	1500
Downtown	L-14	1500
Mianus	L-13	500
Roundhill	L-11	900
Perryride	L-16	1300

Input

σ BRANCH_NAME="perryride" (LOAN)

Output

BRANCH_NAME	LOAN_NO	AMOUNT
Perryride	L-15	1500
Perryride	L-16	1300

Project Operation

- This operation shows the list of those attributes that we wish to appear in the result. Rest of the attributes are eliminated from the table.
- It is denoted by ∏.

Notation: ∏ A1, A2, An (r)

Where

A1, **A2**, **A3** is used as an attribute name of relation **r.**

NAME	STREET	CITY
Jones	Main	Harrison
Smith	North	Rye
Hays	Main	Harrison
Curry	North	Rye
Johnson	Alma	Brooklyn
Brooks	Senator	Brooklyn

Input

∏ NAME, CITY (CUSTOMER)

NAME	CITY
Jones	Harrison
Smith	Rye
Hays	Harrison
Curry	Rye
Johnson	Brooklyn
Brooks	Brooklyn

Cartesian product

- The Cartesian product is used to combine each row in one table with each row in the other table. It is also known as a cross product.
- It is denoted by X.

Notation: E X D

DEPARTMENT

DEPT_NO	DEPT_NAME
A	Marketing
B	Sales
C	Legal

EMPLOYEE

EMP_ID	EMP_NAME	EMP_DEPT
1	Smith	A
2	Harry	C
3	John	B

Input

EMPLOYEE X DEPARTMENT

Output:

EMP_ID	EMP_NAME	EMP_DEPT	DEPT_NO	DEPT_NAME
1	Smith	A	A	Marketing
1	Smith	A	B	Sales
1	Smith	A	C	Legal
2	Harry	C	A	Marketing
2	Harry	C	B	Sales
2	Harry	C	C	Legal
3	John	B	A	Marketing
3	John	B	B	Sales
3	John	B	C	Legal

Rename Operation

The rename operation is used to rename the output relation. It is denoted by **rho** (ρ).

ρ(STUDENT1, STUDENT)

Restriction

Restricts the rows that can be chosen from a relation using a WHERE clause

Takes a horizontal subset of values from the original relation

Example:

```
SELECT * FROM employee WHERE salary > 10000;
```

Projection

Projection is projecting a set of attributes of a relation so that rows of values corresponding to those columns will figure in the output

This takes a **vertical subset** of the relation

Example:

SELECT empid, name, salary **FROM** employee;

Set Operators

Set Operators

- ✓ Union operator retrieves the records from both queries without duplicate.
- ✓ Coulmn heading will be selected from the prior query statement.
- ✓ Union All retrieves all the records from both queries (with duplicate).
- ✓ Intersect operator retrieve the common records from both query statements.
- ✓ Minus operator retrieve the records from first query , the records are not available in second query.

Point to be followed while using SET operators

- ✓ The number of columns must be same in all participating query
- ✓ Column heading will be selected from the first query for displaying the output.
- ✓ Data types of the column list must be match with all the query.
- ✓ UNION and INTERSECT operators are commutative, i.e. the order of queries is not important; it doesn't change the final result.
- ✓ Set operators can be the part of sub queries.
- ✓ Set operators can't be used in SELECT statements containing TABLE collection expressions.
- ✓ For update clause is not allowed with the set operators.

- ✓ Set operators are used to join the results of two (or more) SELECT statement.
- ✓ The following set operators are available in SQL
    - ✓ Union
    - ✓ Union All
    - ✓ Intersect
    - ✓ Minus

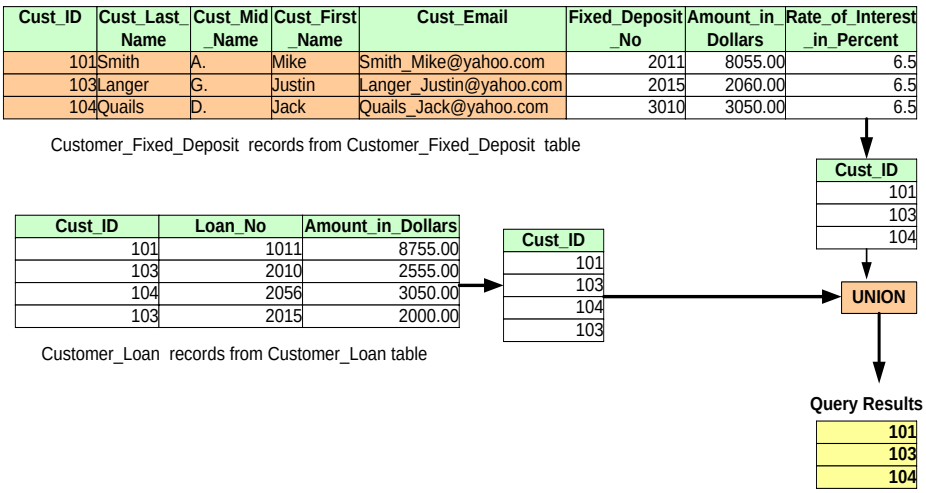
Retrieval using UNION

List all the customer who has either Fixed Deposit or Loan or Both

SELECT Cust_ID FROM Customer_Fixed_Deposit UNION SELECT Cust_ID FROM Customer_Loan;

The UNION operation

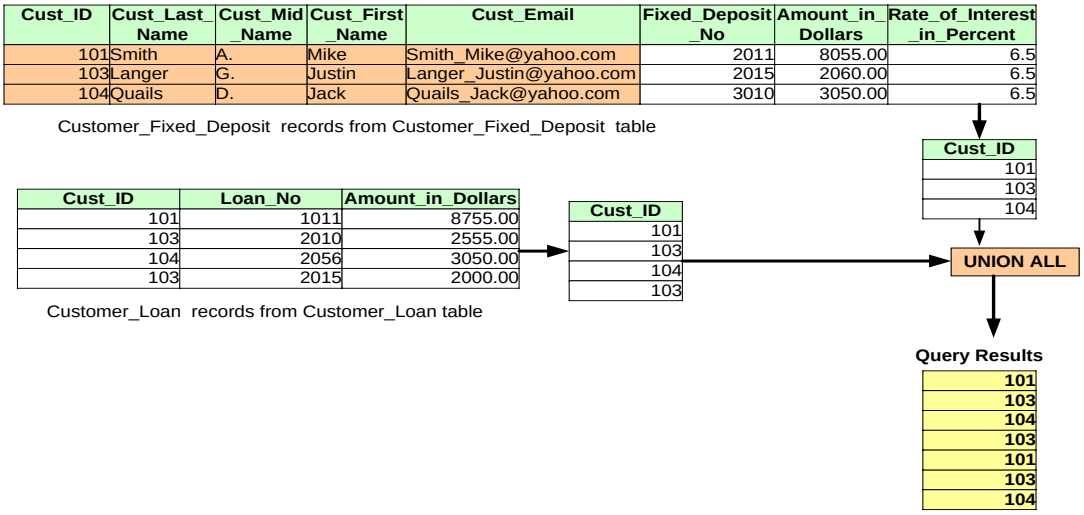
- Combines the rows from two sets of query results.
- By default, the UNION operation eliminates duplicate rows as part of its processing.



Union All

SELECT Cust_ID FROM Customer_Fixed_Deposit UNION ALL

SELECT Cust_ID FROM Customer_Loan;



Union - Restrictions

- ✓ The SELECT statements must contain the **same number of columns**
- ✓ Data type
- ✓ Each column in the first table must be the same as the **data type** of the corresponding column in the second table.
- ✓ Data width and column name can differ

- ✓ Neither of the two tables can be sorted with the ORDER BY clause.
- ✓ Combined query results can be sorted.

Retrieval using INTERSECT

List all the customer who have both Fixed Deposit and Loan.

SELECT Cust_ID FROM Customer_Fixed_Deposit INTERSECT SELECT Cust_ID FROM Customer_Loan;

Minus

Get All the Customer who have not taken loan

SELECT Cust_ID FROM Customer_Account_details MINUS SELECT Cust_Id FROM Customer_loan;

JOINS-INTRODUCTION

Introduction to Joins

Definition

Joins in a database management system (DBMS) are used to combine rows from two or more tables based on a related column between them. It allows for the retrieval of data from multiple tables simultaneously.

Purpose

The primary goal of using joins is to link data across tables, enabling the extraction of meaningful information by establishing relationships between the tables.

Key Point

Joins facilitate the creation of a unified view of data by merging related information from different tables.

Importance of Joins in Database Operations

Data Integration

Joins play a crucial role in integrating data from multiple tables, providing a comprehensive and holistic view of the database.

Efficiency

By using joins, redundant data storage is minimized, leading to more efficient database operations and reduced storage requirements.

Flexibility

Joins offer the flexibility to extract specific data sets by combining information from different tables based on user-defined criteria.

Common Challenges with Joins

Performance Impact

Improper use of joins can lead to performance issues, especially when dealing with large datasets or complex join conditions.

Data Integrity

Incorrect join operations can result in data integrity issues, leading to inaccurate or misleading query results.

Optimization

Optimizing join operations is essential to ensure efficient query execution and minimize resource utilization.

Considerations for Using Joins

Table Relationships

Understanding the relationships between tables is crucial for determining the appropriate type of join to use in different scenarios.

Query Optimization

Employing best practices for join operations can significantly impact query performance and overall database efficiency

Scalability

Considering the scalability of join operations is essential, especially when dealing with growing datasets and evolving database requirements.

Types of Joins

- Cartesian Product
- Inner join
- Equi join
- Outer join
    - Left-outer join
    - Right-outer join
- Self join

Cartesian Product Or Cross Join

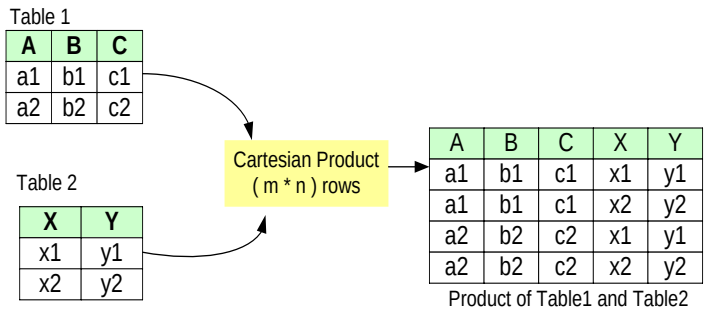
- A Cartesian product, also known as a cross join.
- Type of join in SQL where each row from one table is combined with each row from another table.
- Results in a Cartesian product of the two tables will be quite large if the tables are substantial.

It doesn't require any specific condition to match rows between the tables, unlike other join types.

EXAMPLE:

SELECT * FROM Table A CROSS JOIN TableB;

Cartesian Product Or Cross Join -EXAMPLE



CONDITION JOINS

Introduction

- SQL joins are used to combine rows from two or more tables based on related columns between them.
- Joins allow us to retrieve data from multiple tables, enabling comprehensive analysis.

Definition

- Conditional joins involve specifying additional conditions in the join clause beyond just matching columns.

Purpose

- Used for filtering data based on specific criteria, resulting in more precise data retrieval.

Types of Condition Joins

- Inner Join
- Left Join
- Right Join
- Full Outer Join

Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	XXXXXXXXXX	18
2	PRATIK	BIHAR	XXXXXXXXXX	19
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19

Student Course

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

Inner Join

Returns only the rows that have matching values in both tables.

SELECT StudentCourse.COURSE_ID, Student.NAME, Student.AGE FROM Student

INNER JOIN StudentCourse ON Student.ROLL_NO = StudentCourse.ROLL_NO;

Output:

COURSE_ID	NAME	Age
1	HARSH	18
2	PRATIK	19
2	RIYANKA	20
3	DEEP	18
1	SAPTARHI	19

Left Join

Returns all records from the left table and the matched records from the right table.

SELECT Student.NAME,StudentCourse.COURSE_ID FROM Student LEFT JOIN

StudentCourse ON StudentCourse.ROLL_NO = Student.ROLL_NO;

Output:

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	*NULL*
ROHIT	*NULL*
NIRAJ	*NULL*

Right Join

Returns all records from the right table and the matched records from the left table.

SELECT Student.NAME,StudentCourse.COURSE_ID FROM Student

RIGHT JOIN StudentCourse ON StudentCourse.ROLL_NO = Student.ROLL_NO;

Output:

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
NULL	4
NULL	5
NULL	4

Full Join

Returns all rows from both tables, matching rows from both tables where available, and fills in NULLs for missing matches.

SELECT Student.NAME,StudentCourse.COURSE_ID FROM Student

FULL JOIN StudentCourse ON StudentCourse.ROLL_NO = Student.ROLL_NO;

Output:

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	NULL
ROHIT	NULL
NIRAJ	NULL
NULL	4
NULL	5
NULL	4

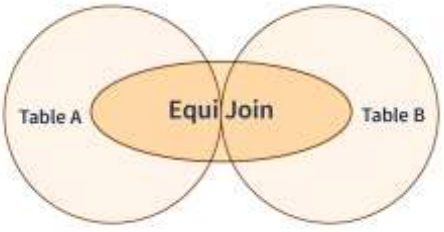
EQUI JOINS

Definition

An equi join is a type of join operation in SQL used to combine tables based on the equality of values in specified columns.

Purpose

Facilitates the retrieval of related data from multiple tables by matching rows with equal values.



Key Characteristics of Equi Join

Matching Column

Join condition is based on a specific column or set of columns shared between the tables.

Equality Operator

Join condition typically utilizes the equality operator (=) to match rows.

Result Set

Includes only the rows where values in the specified columns are equal.

Student

id	name	class	city
3	Hina	3	Delhi
4	Megha	2	Delhi
6	Gouri	2	Delhi

Record

id	class	city
9	3	Delhi
10	2	Delhi
12	2	Delhi

Syntax

SELECT column_list FROM table1, table2....WHERE table1.column_name = table2.column_name;

SELECT student.name, student.id, record.class, record.city FROM student, record WHERE student.city = record.city;

name	id	class	city
Hina	3	3	Delhi
Megha	4	3	Delhi
Gouri	6	3	Delhi
Hina	3	2	Delhi
Megha	4	2	Delhi
Gouri	6	2	Delhi
Hina	3	2	Delhi
Megha	4	2	Delhi
Gouri	6	2	Delhi

NON EQUI JOIN

- NON EQUI JOIN performs a JOIN using comparison operator other than equal(=) sign like >, <, >=, <= with conditions.

Syntax

SELECT * FROM table_name1, table_name2 WHERE table_name1.column [> | < | >= | <=] table_name2.column;

SELECT student.name, record.id, record.city FROM student, record WHERE Student.id < Record.id ;

name	id	city
Hina	9	Delhi
Megha	9	Delhi
Gouri	9	Delhi
Hina	10	Delhi
Megha	10	Delhi
Gouri	10	Delhi
Hina	12	Delhi
Megha	12	Delhi
Gouri	12	Delhi

Best Practices

Choose Relevant Columns

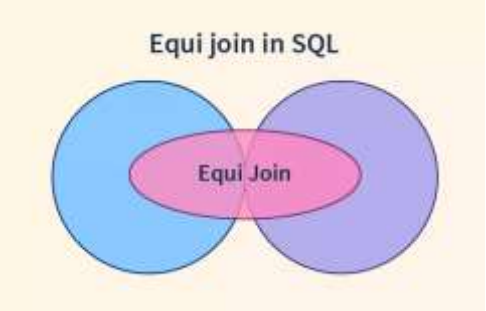
- Select columns that represent meaningful relationships between tables.

Ensure Data Integrity

- Verify that join columns have consistent data types and values.

Optimize Performance

- Use indexes on join columns to improve query performance.



NATURAL JOINS

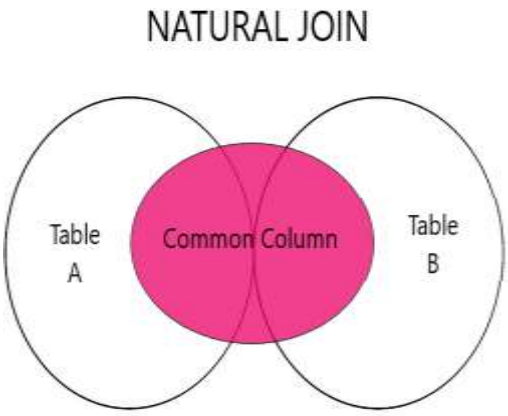
Introduction to Natural Joins

Definition

A natural join in SQL combines rows from two or more tables based on the common column(s) between them. It automatically selects the columns with the same name in both tables.

Key Point

Natural joins simplify the process of joining tables by automatically matching columns with the same name.



How Natural Joins Work

Automatic Matching

Natural joins compare all columns of two tables that have the same column name, and the resulting joined table contains those columns once

Simplification

This type of join simplifies the process of writing SQL queries by eliminating the need to specify the columns to join on.

Common Usage

Natural joins are often used when the tables being joined have columns with the same name and data type.

Considerations for Natural Joins

Column Naming Conventions

Using natural joins requires careful consideration of column names to ensure that the join produces the desired results.

Impact on Query Performance

Natural joins can impact the performance of SQL queries, especially when dealing with large datasets and complex joins.

Best Practices

Adhering to best practices for column naming and understanding the implications of natural joins is essential for effective usage.

Features of Natural Join

- It will perform the Cartesian product.
- It finds consistent tuples and deletes inconsistent tuples.
- Then it deletes the duplicate attributes.
- They reduce the complexity of queries by eliminating the need to explicitly specify the columns to join on.

Employee

EMP_ID	EMP_NAME	DEPT_NAME
1	SUMIT	HR
2	JOEL	IT
3	BISWA	MARKETING
4	VAIBHAV	IT
5	SAGAR	SALES

Department

DEPT_NAME	MANAGER_NAME
IT	ROHAN
SALES	RAHUL
HR	TANMAY
FINANCE	ASHISH
MARKETING	SAMAY

SELECT * FROM employee NATURAL JOIN department;

EMP_ID	EMP_NAME	DEPT_NAME	MANAGER_NAME
1	SUMIT	HR	TANMAY
2	JOEL	IT	ROHAN
3	BISWA	MARKETING	SAMAY
4	VAIBHAV	IT	ROHAN
5	SAGAR	SALES	RAHUL

NATURAL JOIN Vs INNER JOIN

- Both NATURAL JOIN and INNER JOIN produce the same result in this example because they are matching on the same columns with the same values.
- NATURAL JOIN implicitly matches on all columns with the same name, which may not always be desired and can lead to unexpected results if there are other columns with the same name in the tables.
- INNER JOIN, on the other hand, allows for explicit control over the join conditions and is generally considered safer and more predictable.

Simple Join and Division

Simple Join

- ✓ For the below query the output is cartesian prodcut

  Sql> select * from emp,dept;

- ✓ For all the records in the first table (emp) , Each and every record in the second table (dept) will be executed.
- ✓ That is , 14 x 4 Records will be in output.

Syntax : where <table_name1>.<column_name>

=<table_name2>.<column_name>

Example 1:

```
select * from emp,dept where emp.deptno= dept.deptno;
```

dept Table:

deptno	dname	loc
10	Accounting	New York
20	Research	Dallas
30	Sales	Chicago
40	Marketing	Los Angeles

emp Table:

empno	ename	deptno
7369	Smith	20
7499	Allen	30
7521	Ward	30
7566	Jones	20

7698	Blake	30

OUTPUT

empno	ename	deptno	deptno_1	dname	loc
7369	Smith	20	20	Research	Dallas
7499	Allen	30	30	Sales	Chicago
7521	Ward	30	30	Sales	Chicago
7566	Jones	20	20	Research	Dallas
7698	Blake	30	30	Sales	Chicago

- The emp.deptno and dept.deptno columns are matched where they have the same value.
- Rows from both tables where emp.deptno equals dept.deptno are combined into one row in the result table.
- Columns from both tables are retained in the result table. To avoid ambiguity, the duplicate columns (deptno in this case) are suffixed with _1.

Example 2:

- select ename,dname from emp,dept where emp.deptno= dept.deptno;

ename	dname
Smith	Research
Allen	Sales
Ward	Sales
Jones	Research
Blake	Sales

- The emp.deptno and dept.deptno columns are matched where they have the same value.
- Rows from both tables where emp.deptno equals dept.deptno are combined into one row in the result table.
- Only the selected columns (ename and dname) are retained in the result table.

Introduction to Division Operator

The division operator in SQL is a fundamental arithmetic operator used to divide one number by another.

It is represented by the symbol / and is used to perform division operations in SQL queries.

When using the division operator with integers, it's important to understand how SQL handles the result, particularly in cases where the result is a non-integer value.

Handling Division with SQL

Dealing with Remainders

Unlike some other programming languages, SQL handles division differently, especially when dealing with integer division and the treatment of remainders.

Division by Zero

Exploring the behavior of SQL when attempting to divide by zero, including potential errors and how to handle such scenarios.

Purpose

It helps to identify records that are related to all values in another set.