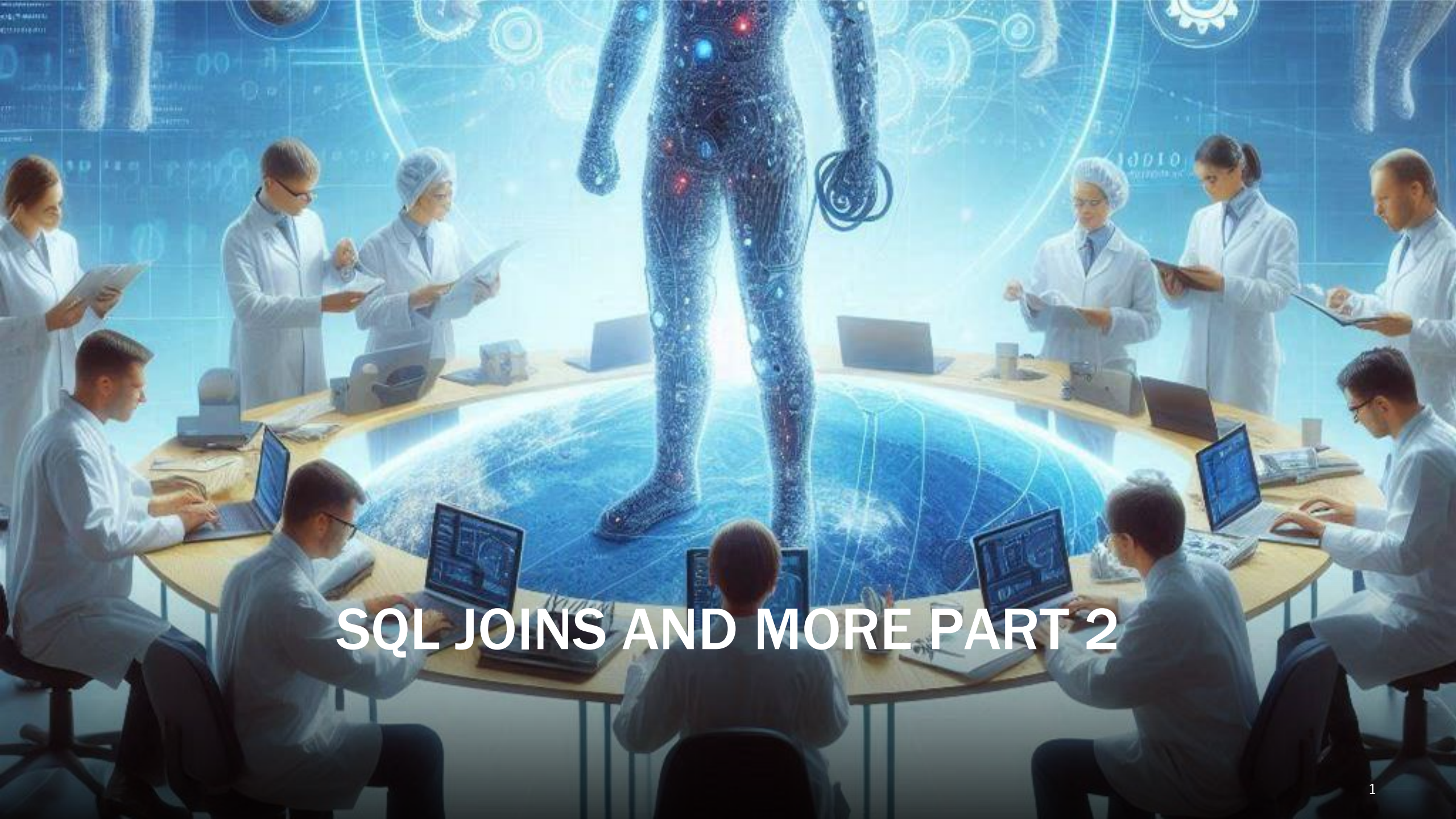


SQL JOINS AND MORE PART 2

SQL REVIEW

- SELECT
- WHERE
- CREATE TABLE
- PRIMARY KEY
- FOREIGN KEY
- REFERENCES
- UPDATE
- SET
- INSERT INTO
- VALUES
- DROP TABLE
- TRUNCATE TABLE
- IN
- BETWEEN
- AS
- COUNT(*)
- SUM()
- AVG(),
- MAX(),
- MIN()
- GROUP BY
- HAVING
- ORDER BY
- DISTINCT
- LIKE
- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL OUTER JOIN
- ABS()
- LOG()
- RAND()
- ROUND()
- POW()
- SQRT()
- CONCAT()
- LOWER()
- UPPER()
- TRIM()
- SUBSTRING()
- CURDATE()
- NOW()
- TIME()
- DATEDIFF()
- TIMEDIFF()
- REPLACE()

BYU

DISTINCT

- The SQL DISTINCT clause is used to remove duplicate values from a result set.
- When DISTINCT is applied to a column in a SELECT statement, it ensures that only unique values for that column are returned, eliminating any duplicates.

Syntax:

```
SELECT DISTINCT column1, column2, ...
FROM table_name;
```

Get a list of unique customer first names:

```
SELECT DISTINCT FirstName
FROM Customer;
```

Retrieve unique invoice dates:

This query fetches all the unique dates on which invoices were issued from the Invoice table. If multiple invoices were issued on the same day, the date will appear only once in the result set.

```
SELECT DISTINCT InvoiceDate
FROM Invoice;
```

When to Use DISTINCT:

- **To eliminate duplicates** from your result set when you're only interested in unique values.
- **On columns with repeating values**, where you want to count or list only unique occurrences.

BYU

FUNCTIONS IN SQL

function / argument

A function operates on an expression enclosed in parentheses, called an argument, and returns a value. Usually, the argument is a simple expression, such as a column name or fixed value. Some functions have several arguments, separated by commas, and a few have no arguments at all.

Function / Argument

- operates on an expression enclosed in parentheses(argument) returning a value

- Numeric
- String
- Date/Time

BYU

NUMERIC FUNCTIONS

Function	Description	Example
ABS(n)	Returns the absolute value of *n*	`SELECT ABS(-5);` returns 5
LOG(n)	Returns the natural logarithm of *n*	`SELECT LOG(10);` returns 2.302585092994046
POW(x, y)	Returns *x* to the power of *y*	`SELECT POW(2, 3);` returns 8
RAND()	Returns a random number between 0 (inclusive) and 1 (exclusive)	`SELECT RAND();` returns 0.11831825703225868
ROUND(n, d)	Returns *n* rounded to *d* decimal places	`SELECT ROUND(16.25, 1);` returns 16.3
SQRT(n)	Returns the square root of *n*	`SELECT SQRT(25);` returns 5

BYU

COMBINING FUNCTIONS:

You can combine these functions for more complex calculations. For example, you could round the result of a logarithm calculation:

Example: Rounding the logarithm of invoice totals

```
SELECT InvoiceID, ROUND(LOG(Total), 2) AS RoundedLogTotal
FROM Invoice
WHERE Total > 0;
```

Explanation: This query calculates the logarithm of the invoice total and rounds the result to 2 decimal places.

BYU

STRING FUNCTIONS

Function	Description	Example
CONCAT(s1, s2, ...)	Returns the string that results from concatenating the string arguments	`SELECT CONCAT('Dis', 'en', 'gage');` returns 'Disengage'
LOWER(s)	Returns the lowercase *s*	`SELECT LOWER('MySQL');` returns 'mysql'
REPLACE(s, from, to)	Returns the string *s* with all occurrences of *from* replaced with *to*	`SELECT REPLACE('This and that', 'and', 'or');` returns 'This or that'
SUBSTRING(s, pos, len)	Returns the substring from *s* that starts at position *pos* and has length *len*	`SELECT SUBSTRING('Boomerang', 1, 4);` returns 'Boom'
TRIM(s)	Returns the string *s* without leading and trailing spaces	`SELECT TRIM('  test  ');` returns 'test'
UPPER(s)	Returns the uppercase *s*	`SELECT UPPER('mysql');` returns 'MYSQL'

BYU

STRING FUNCTION EXAMPLES: CONCAT(), LOWER()

CONCAT(): Concatenating Strings

The CONCAT() function combines two or more strings into one. It's useful for creating full names from separate FirstName and LastName columns, for example.

Example: Combining first and last names into a full name

```
SELECT CustomerID, CONCAT(FirstName, ' ', LastName) AS FullName
FROM VIPCustomer;
```

LOWER(): Converting Text to Lowercase

The LOWER() function converts all characters in a string to lowercase. This can be useful when comparing text data case-insensitively.

Example: Converting email addresses to lowercase

```
SELECT CustomerID, LOWER(Email) AS LowercaseEmail
FROM VIPCustomer;
```

BYU

STRING FUNCTION EXAMPLES: TRIM() AND SUBSTRING()

TRIM(): Removing Leading and Trailing Spaces

The TRIM() function removes any leading and trailing spaces from a string. It's useful for cleaning up user input or inconsistent data.

Example: Removing extra spaces from

```
SELECT CustomerID, TRIM(FirstName) AS CleanedFirstName,
TRIM(LastName) AS CleanedLastName
FROM VIPCustomer;
```

Explanation: This query removes any extra spaces at the beginning or end of the FirstName and LastName fields.

SUBSTRING(): Extracting Part of a String

The SUBSTRING() function extracts a portion of a string, starting from a given position and extracting a specified number of characters.

Example: Extracting the first 3 characters of the first name

```
SELECT CustomerID, SUBSTRING(FirstName, 1, 3)
AS NamePrefix
FROM VIPCustomer;
```

Explanation: This query extracts the first 3 characters from the FirstName of each customer, which can be useful for generating a prefix or performing partial matches.

COMBINING STRING FUNCTIONS

You can also combine these functions for more complex string manipulations.

Example: Concatenating a cleaned full name and lowercased email address

```
SELECT CustomerID, CONCAT(TRIM(FirstName), ' ', TRIM(LastName)) AS FullName,
LOWER(Email) AS LowercaseEmail
FROM VIPCustomer;
```

Explanation: This query trims any extra spaces from FirstName and LastName, concatenates them into a full name, and converts the Email to lowercase.

BYU

LIKE

The SQL LIKE clause is used in a WHERE statement to search for a specified pattern in a column. It's commonly used with string data types to find rows where a column matches a particular pattern. Wildcards are often used with LIKE to specify patterns:

%: Represents zero or more characters.
_: Represents a single character.

Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE column LIKE pattern;
```

Let's say we have a Customer table with the following columns: CustomerID, FirstName, LastName, and Email. We can use the LIKE clause to perform different types of pattern matching on this data.

Find customers with first names that start with 'A':

```
SELECT CustomerID, FirstName, LastName
FROM Customer
WHERE FirstName LIKE 'A%';
```

Find customers with a last name that starts with 'S' and has 5 letters:

```
SELECT CustomerID, FirstName, LastName
FROM Customer
WHERE LastName LIKE 'S____';
```

Find customers whose email starts with 'a' and ends with '.com':

```
SELECT CustomerID, FirstName, LastName, Email
FROM Customer
WHERE Email LIKE 'a%@%.com';
```

BYU

DATE/TIME FUNCTIONS

Function	Description	Example
CURDATE() ***CURTIME()*** ***NOW()***	Returns the current date, time, or date and time in `'YYYY-MM-DD'`, `'HH:MM:SS'`, or `'YYYY-MM-DD HH:MM:SS'` format	`SELECT CURDATE();` returns '2019-01-25' `SELECT CURTIME();` returns '21:05:44' `SELECT NOW();` returns '2019-01-25 21:05:44'

DATE(expr)
TIME(expr)
	Extracts the date or time from a date or datetime expression *expr*	`SELECT DATE('2013-03-25 22:11:45');`
		returns '2013-03-25'
		`SELECT TIME('2013-03-25 22:11:45');`
		returns '22:11:45'

DAY(d) ***MONTH(d)*** ***YEAR(d)***	Returns the day, month, or year from date *d*	`SELECT DAY('2016-10-25');` returns 25 `SELECT MONTH('2016-10-25');` returns 10 `SELECT YEAR('2016-10-25');` returns 2016
HOUR(t) ***MINUTE(t)*** ***SECOND(t)***	Returns the hour, minute, or second from time *t*	`SELECT HOUR('22:11:45');` returns 22 `SELECT MINUTE('22:11:45');` returns 11 `SELECT SECOND('22:11:45');` returns 45

DATEDIFF(expr1, expr2)
TIMEDIFF(expr1, expr2)
	Returns *expr1* - *expr2* in number of days or time values, given *expr1* and *expr2* are date, time, or datetime values	`SELECT DATEDIFF('2013-03-10', '2013-03-04');`
		returns 6
		`SELECT TIMEDIFF('10:00:00', '09:45:30');`
		returns 00:14:30

BYU

CURTIME(): RETRIEVES THE CURRENT TIME

The CURTIME() function returns the current time (without the date) in the format HH:MM:SS or HHMMSS depending on the query context.

Example: Retrieve the current time and check which invoices were created today at that time

```
SELECT InvoiceID, Total, CURTIME() AS CurrentTime
FROM Invoice
WHERE CURDATE() = InvoiceDate;
```

Explanation: This query shows the InvoiceID and Total of invoices created today, along with the current time. It uses CURTIME() to retrieve the current time and CURDATE() to compare only the date part of the InvoiceDate.

BYU

NOW(): RETRIEVES THE CURRENT DATE AND TIME

The NOW() function returns the current date and time in the format YYYY-MM-DD HH:MM:SS. It's useful when you need to log timestamps or compare data based on both date and time.

Example: Retrieve all invoices created in the last hour

```
SELECT InvoiceID, Total, InvoiceDate
FROM Invoice
WHERE InvoiceDate >= NOW() - INTERVAL 1 HOUR;
```

Explanation: This query retrieves all invoices created in the last hour. It uses the NOW() function to get the current date and time and compares it with the InvoiceDate column to filter invoices created within the last hour.

BYU

DATEDIFF(): CALCULATES THE DIFFERENCE BETWEEN TWO DATES

The DATEDIFF() function returns the difference in days between two dates. It's commonly used to calculate time intervals, such as how long a customer has been a VIP or the number of days since an invoice was issued.

Example: Calculate how many days have passed since each customer joined

```
SELECT CustomerID, FirstName, LastName,
DATEDIFF(NOW(), JoinDate) AS DaysSinceJoined
FROM VIPCustomer;
```

Explanation: This query calculates the number of days since each customer joined the VIP program by subtracting the JoinDate from the current date and time (NOW()).

Example 2: Find overdue invoices (overdue by more than 30 days)

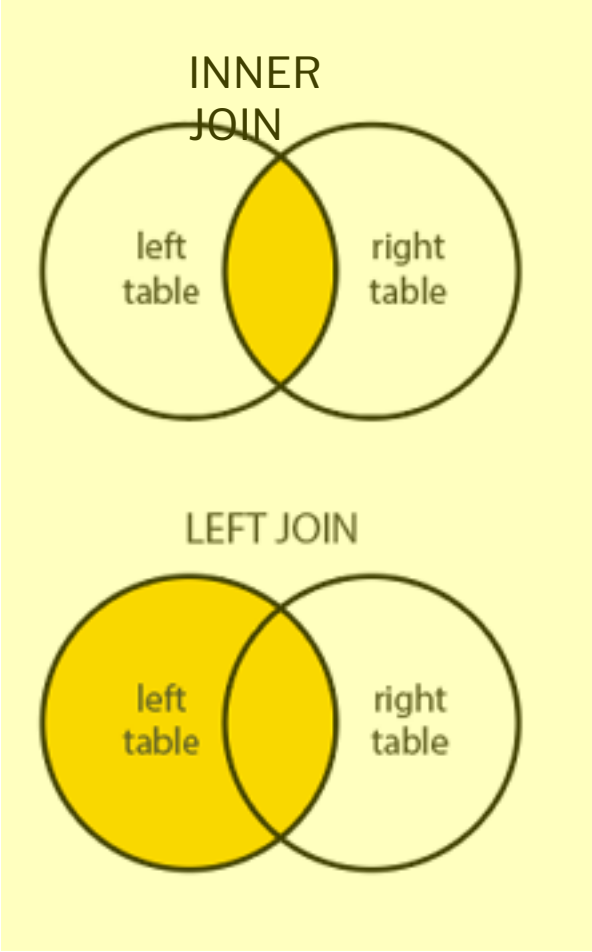
```
SELECT InvoiceID, CustomerID,
Total, DATEDIFF(NOW(),
InvoiceDate) AS DaysOverdue
FROM Invoice
WHERE DATEDIFF(NOW(),
InvoiceDate) > 30;
```

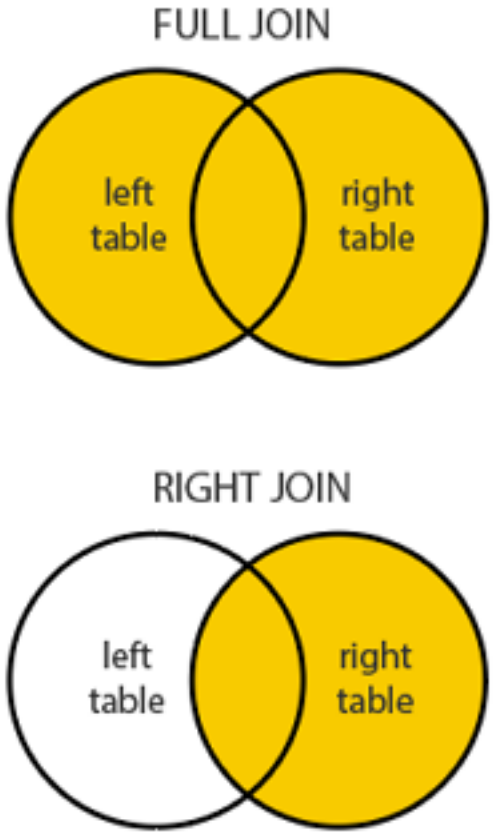
Explanation: This query identifies invoices that are overdue by more than 30 days by calculating the difference between the current date (NOW()) and the InvoiceDate.

BYU

JOINING TABLES

- **JOIN** is used to combine rows from two or more tables based on a related column between them.
- There 4 major types of joins.
  - **INNER JOIN** returns rows when there is a match in both tables.
  - **LEFT JOIN** returns all rows from the left table and the matched rows from the right table
  - **RIGHT JOIN** returns all rows from the right table and the matched rows from the left table
  - **FULL (OUTER) JOIN** returns all rows when there is a match in either table. If there is no match





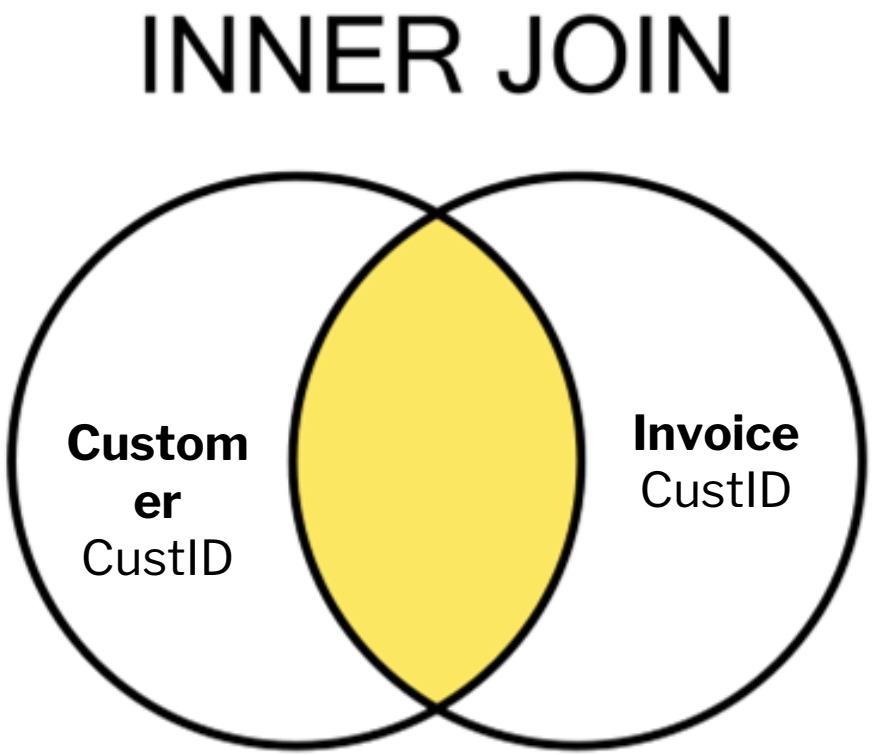
BYU

INNER JOIN

- An INNER JOIN returns rows when there is a match in both tables. It excludes rows where the condition is not met.

Retrieve customer names and their invoice totals by joining Customer and Invoice

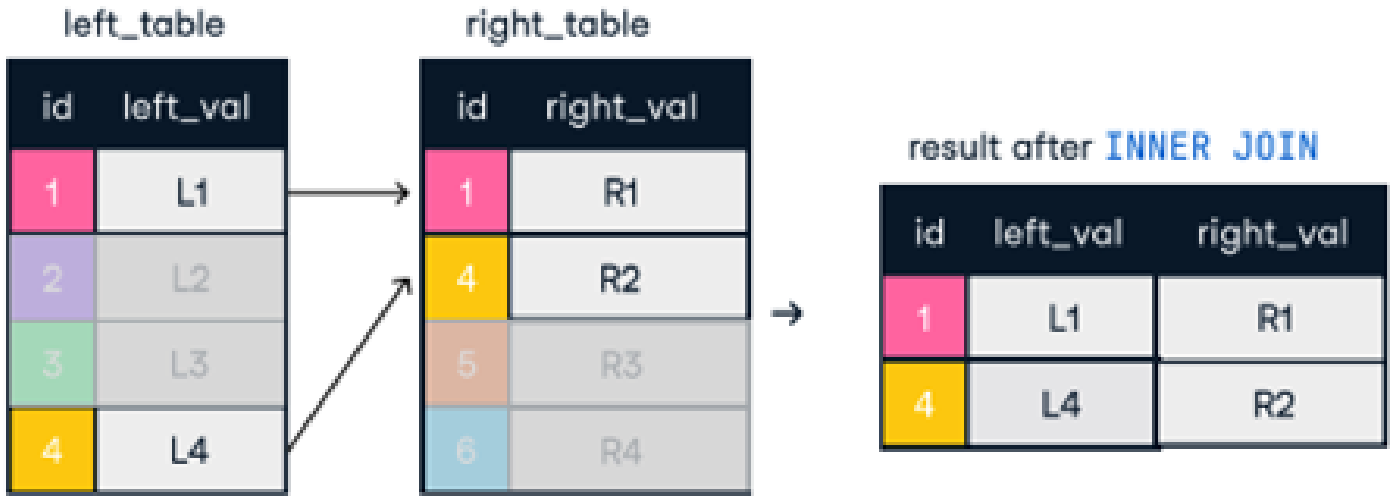
```
SELECT c.FirstName, c.LastName, i.Total
FROM Customer c
INNER JOIN Invoice i ON c.CustomerId =
i.CustomerId;
```



BYU

INNER JOIN

An inner join between two tables will return only records where a joining field, such as a key, finds a match in both tables.

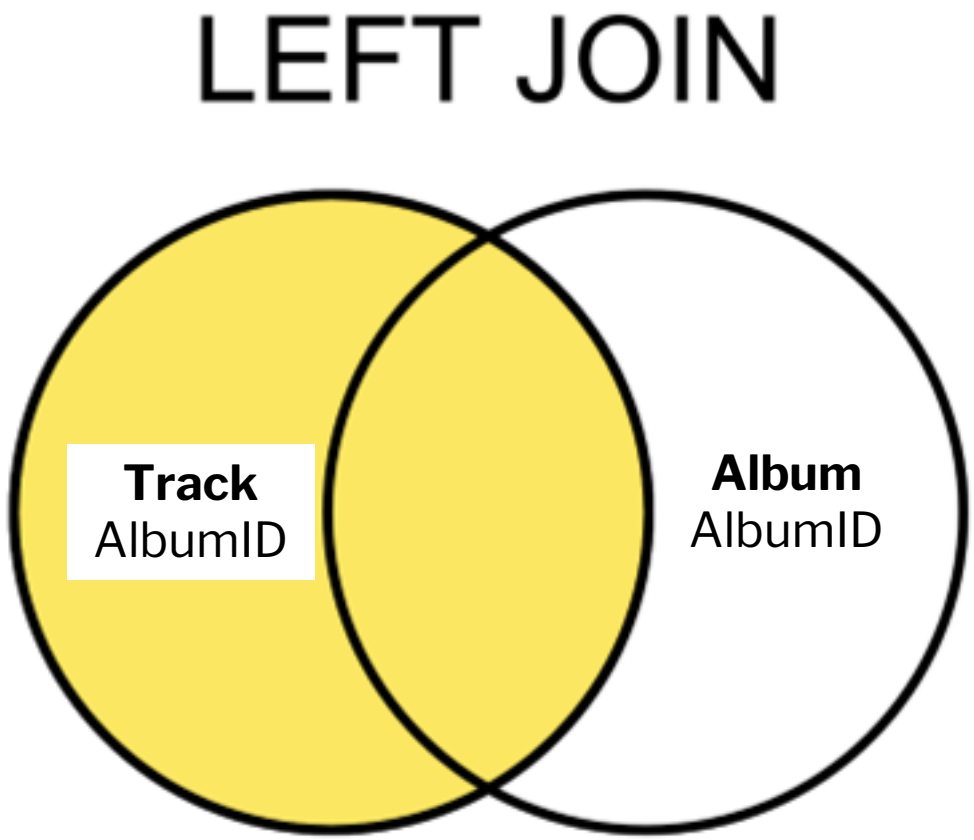


LEFT JOIN

A LEFT JOIN returns all rows from the left table and the matched rows from the right table. If there is no match, the result will include NULL values for the columns from the right table.

List all tracks and their corresponding album titles, including tracks without an album assigned:

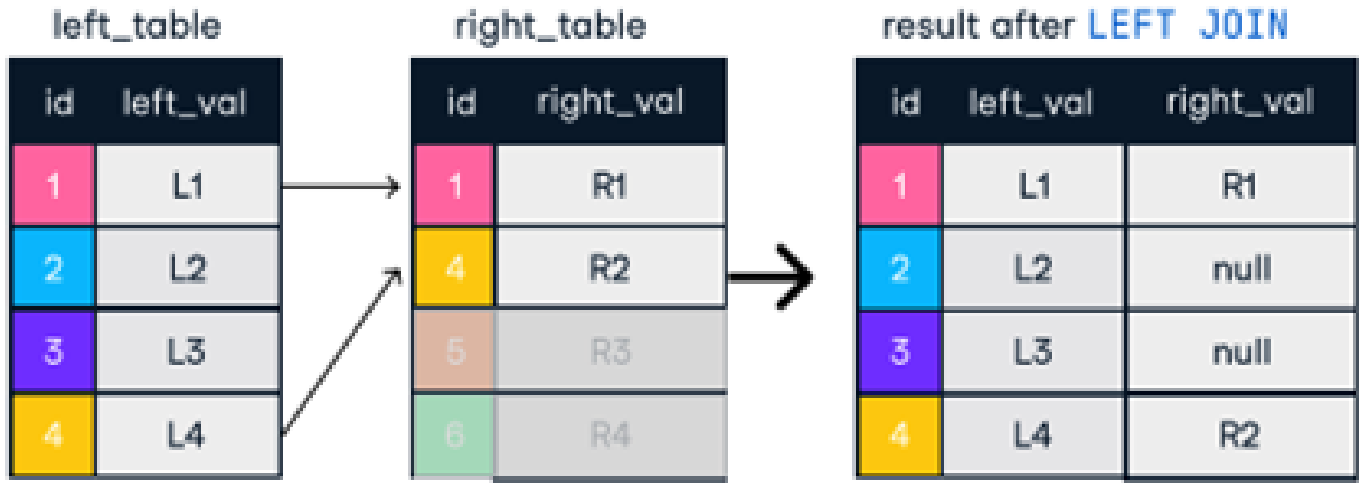
```
SELECT t.Name, a.Title
FROM Track t
LEFT JOIN Album a ON t.AlbumId = a.AlbumId;
```



BYU

LEFT JOIN

A left join keeps all of the original records in the left table and returns missing values for any columns from the right table where the joining field did not find a match.

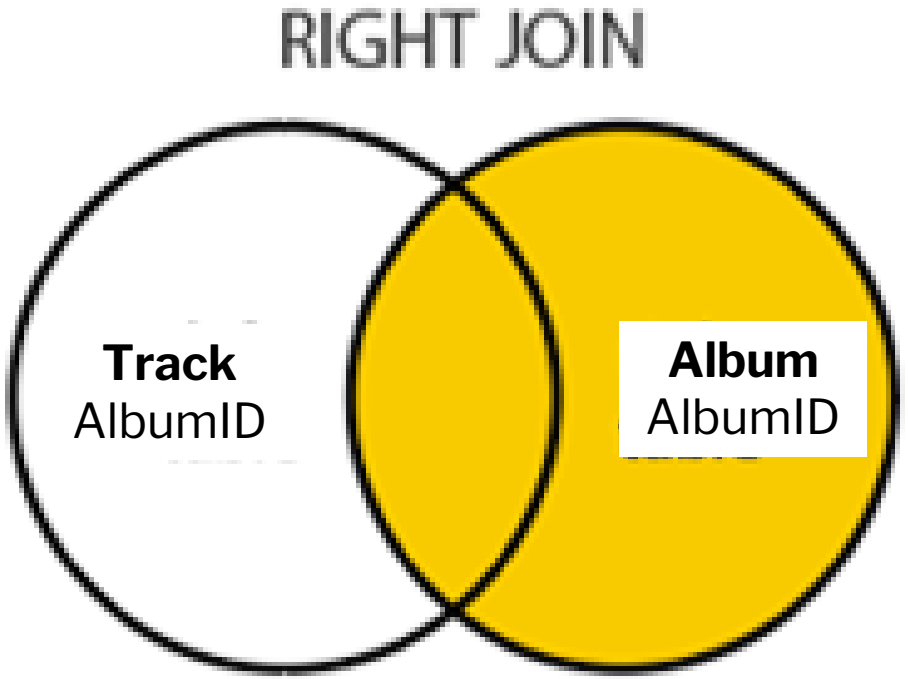


RIGHT JOIN

A RIGHT JOIN returns all rows from the right table and the matched rows from the left table. If there's no match, the result will include NULL values for the left table.

List all albums and the tracks they contain, including albums without tracks

```
SELECT a.Title, t.Name
FROM Track t
RIGHT JOIN Album a ON t.AlbumId = a.AlbumId;
```

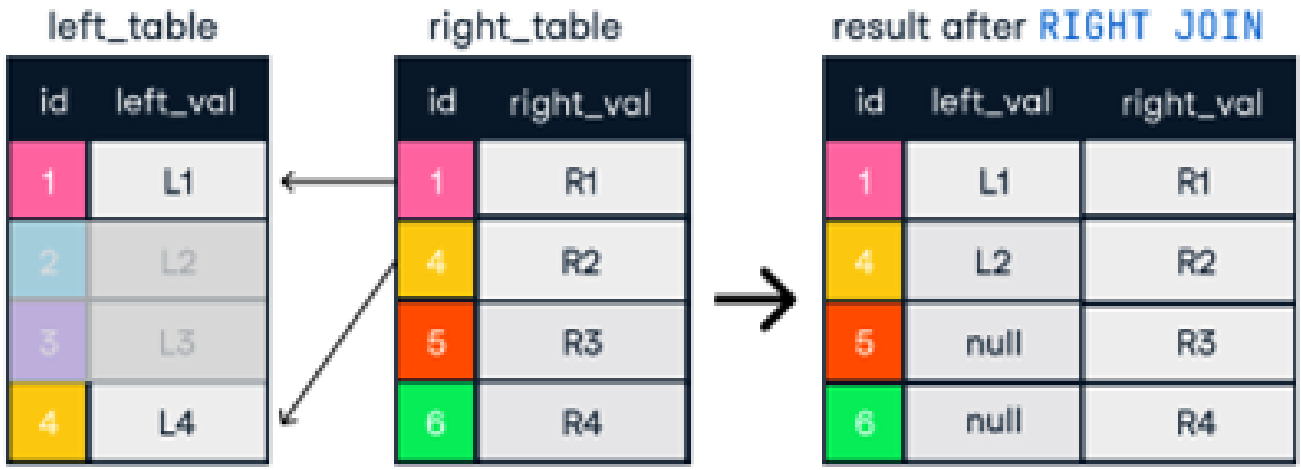


Can be converted into a LEFT JOIN by reversing the tables

BYU

RIGHT JOIN

A right join keeps all of the original records in the right table and returns missing values for any columns from the left table where the joining field did not find a match. Right joins are far less common than left joins, because right joins can always be re-written as left joins.

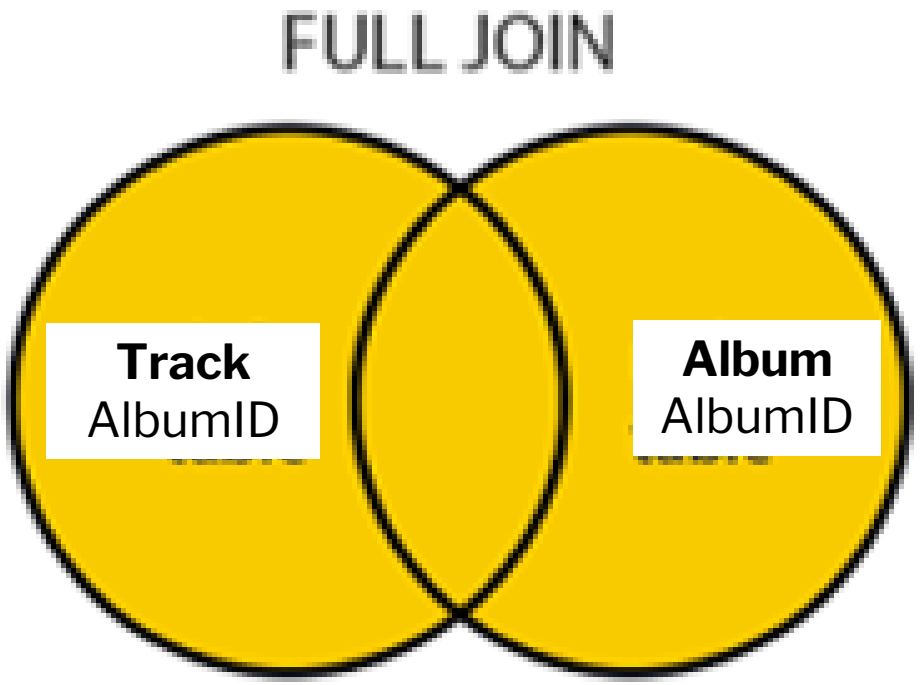


FULL JOIN (AKA FULL OUTER JOIN)

A FULL OUTER JOIN returns all rows when there is a match in either table. If there is no match, the result will include NULL values for the missing data from both tables.

Retrieve all artists and albums, showing nulls where there's no match:

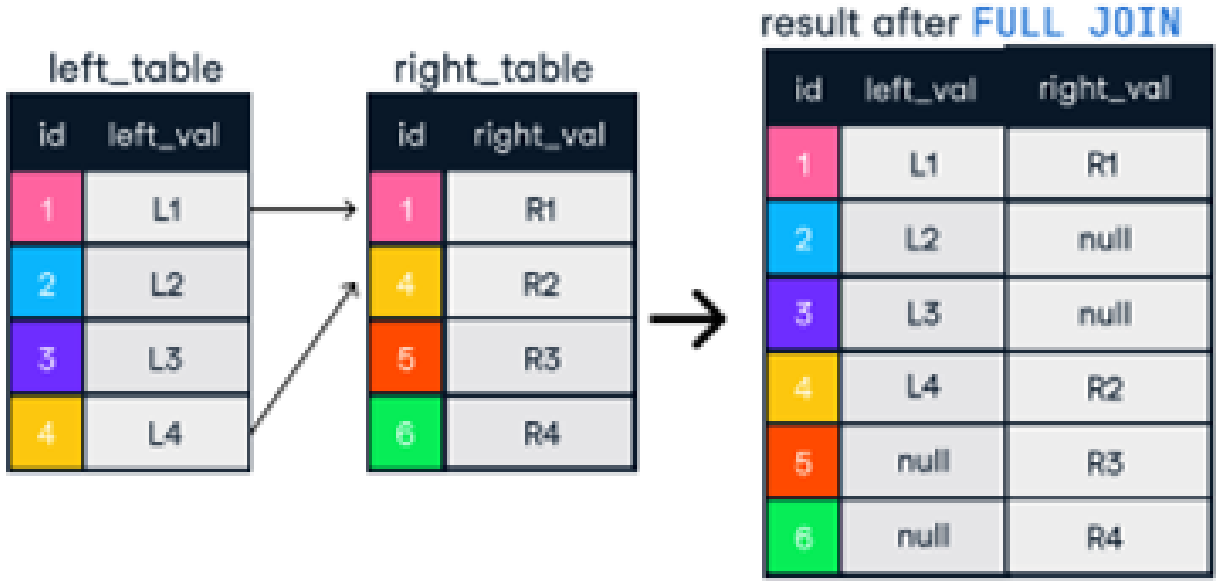
```
SELECT ar.Name AS ArtistName, al.Title AS
AlbumTitle
FROM Artist ar
FULL OUTER JOIN Album al ON ar.ArtistId =
al.ArtistId;
```



BYU

FULL JOIN

A full join combines a left join and right join. A full join will return all records from a table, irrespective of whether there is a match on the joining field in the other table, returning null values accordingly.



UNION

- The UNION operator is used to combine results of two or more SELECT queries into a single result set.
- It removes duplicate rows by default, so the result set contains only unique rows.
- If you want to include all duplicate rows, you can use UNION ALL.

Key Points:

- **Queries must have the same number of columns**.
- **The columns must have compatible data types** in corresponding positions.
- The **column names in the result** will be based on the first SELECT statement.
- By default, UNION removes duplicates, but UNION ALL keeps all rows.

Syntax:

```
SELECT column1, column2, ...
FROM table1
UNION
SELECT column1, column2, ...
FROM table2;
```

Let's say we have two tables: VIPCustomer (for VIP customers) and RegularCustomer (for regular customers). We want to retrieve a list of all customers, combining both tables.

```
SELECT CustomerID, FirstName, LastName
FROM VIPCustomer
UNION
SELECT CustomerID, FirstName, LastName
FROM RegularCustomer;
```

BYU

UNION

The `UNION` operator is used to vertically combine the results of two `SELECT` statements. For `UNION` to work without errors, all `SELECT` statements must have the same number of columns and corresponding columns must have the same data type. `UNION` does not return duplicates.