Query processing order

INTRODUCTION TO ORACLE SQL

Hadrien Lacroix
Content Developer

Why does processing order matter?

- Optimize your queries
  - No unwanted results
  - Faster execution

Example

```
SELECT BillingCountry, AVG(Total) > 100) AS Average
FROM Invoice
WHERE BillingCity <> 'Paris'
GROUP BY BillingCountry
HAVING AVG(Total) > 100
ORDER BY Average DESC
```

Example

```
    SELECT BillingCountry, AVG(Total) > 100) AS Average
->  FROM Invoice
    WHERE BillingCity <> 'Paris'
    GROUP BY BillingCountry
    HAVING AVG(Total) > 100
    ORDER BY Average DESC
```

Example

```
    SELECT BillingCountry, AVG(Total) > 100) AS Average
    FROM Invoice
->  WHERE BillingCity <> 'Paris'
    GROUP BY BillingCountry
    HAVING AVG(Total) > 100
    ORDER BY Average DESC
```

Example

```
    SELECT BillingCountry, AVG(Total) > 100) AS Average
    FROM Invoice
    WHERE BillingCity <> 'Paris'
->  GROUP BY BillingCountry
    HAVING AVG(Total) > 100
    ORDER BY Average DESC
```

Example

```
    SELECT BillingCountry, AVG(Total) > 100) AS Average
    FROM Invoice
    WHERE BillingCity <> 'Paris'
    GROUP BY BillingCountry
->  HAVING AVG(Total) > 100
    ORDER BY Average DESC
```

Example

```
->   SELECT BillingCountry, AVG(Total) > 100) AS Average
     FROM Invoice
     WHERE BillingCity <> 'Paris'
     GROUP BY BillingCountry
     HAVING AVG(Total) > 100
     ORDER BY Average DESC
```

Example

```
    SELECT BillingCountry, AVG(Total) > 100) AS Average
    FROM Invoice
    WHERE BillingCity <> 'Paris'
    GROUP BY BillingCountry
    HAVING AVG(Total) > 100
->  ORDER BY Average DESC
```

What could go wrong?

```
SELECT BillingCountry,
       AVG(Total) > 100) AS Average
FROM Invoice
WHERE BillingCity <> 'Paris'
GROUP BY BillingCountry
HAVING AVG(Total) > 100
ORDER BY Average DESC
```

- Aliases **can't** be used in `WHERE` , `GROUP BY` , and `HAVING`
- Aliases **can** be used in `ORDER BY`

What could go wrong?

```
SELECT BillingCountry,
       AVG(Total) > 100) AS Average
FROM Invoice
WHERE BillingCity <> 'Paris'
GROUP BY BillingCountry
HAVING AVG(Total) > 100
ORDER BY Average DESC
```

- Aggregated values **can't** be filtered out in the `WHERE` clause
- Aggregated values **can** be filtered out in the `HAVING` clause

What could go wrong?

```sql
SELECT BillingCountry,
       AVG(Total) > 100) AS Average
FROM Invoice
WHERE BillingCity <> 'Paris'
GROUP BY BillingCountry
HAVING AVG(Total) > 100)
ORDER BY Average DESC
```

- Single rows **can't** be filtered out in the `HAVING` clause
- Single rows **can** be filtered out in the `WHERE` clause

Query order of execution

1. `FROM` and `JOIN` s: determine which data is being queried
2. `WHERE` : filter individual rows
3. `GROUP BY` : group rows
4. `HAVING` : filter groups
5. `SELECT` : select columns and apply functions on columns
6. `DISTINCT` : remove duplicates
7. `UNION` , `UNION ALL` , `INTERSECT` , `MINUS` : apply set operators
8. `ORDER BY` : order rows

Let's practice!

INTRODUCTION TO ORACLE SQL

Customizing output

INTRODUCTION TO ORACLE SQL

Hadrien Lacroix
Content Developer

Functions

Functions can be used for:

- calculation
- formating
- manipulation
- conversion between data types

Functions and data types

	Numeric data	Character data	Date data
`AVG`	x		
`SUM`	x		
`MIN`	x	x	x
`COUNT`	x	x	x

Data types define what type of data a column can contain.

Types of functions

- **Character functions**
  - *Input*: character values
  - *Outputs*: character, numeric, date values
- **Number functions**
  - *Input*: numeric values
  - *Outputs*: numeric values

- **Date functions**
- **General functions**
- **Conversion functions**

Case manipulation: upper case

`UPPER(column)` : converts all alpha character values to uppercase

```
SELECT UPPER(State) AS State, UPPER(PostalCode) AS PostalCode
FROM Customer
```

```
| State | PostalCode |
|-------|------------|
| CA    | 95014      |
| DF    | 71020-677  |
| AB    | T6G 2C7    |
| BC    | V6C 1G8    |
| QC    | H2G 1A7    |
| ...   |            |
```

Case manipulation: lowercase

`LOWER(column)` : converts all alpha character values to lowercase

```
SELECT LOWER(Email) AS LowercaseEmail
FROM Customer
```

```
| LowercaseEmail            |
|--------------------------|
| luisg@embraer.com.br     |
| leonekohler@surfeu.de    |
| ftremblay@gmail.com      |
| bjorn.hansen@yahoo.no    |
| ...                      |
```

Getting a substring

`SUBSTR(column, m, n)` : returns a portion of a string from position `m` , `n` characters long

```
SELECT Phone
FROM Customer
```

```
| Phone              |
|--------------------|
| +56 02 635 4444    |
| +91 0124 39883988 |
| +44 0131 315 3300 |
```

Goal: Get the country code of a telephone number without `+`

Getting a substring

Goal: get the country code of a telephone number without `+`

```sql
SELECT Phone, SUBSTR(Phone, 2, 2) AS cc
FROM Customer
```

```
| Phone              | cc |
|--------------------|----|
| +56 (0)2 635 4444 | 56 |
| +91 0124 39883988 | 91 |
| +44 0131 315 3300 | 44 |
| +39 06 39733434   | 39 |
| +48 22 828 37 39  | 48 |
| ...                |    |
```

Nested functions

Goal: Generate usernames for customers from first 5 letters of their last name and their id

```
SELECT LastName, CustomerId, CONCAT(SUBSTR(LastName,1,5), CustomerId) AS UserName
FROM customer
```

```
| LastName | CustomerId | UserName |
|----------|------------|----------|
| Almeida  | 12         | Almei12  |
| Barnett  | 28         | Barne28  |
| Bernard  | 39         | Berna39  |
| Brooks   | 18         | Brook18  |
| Brown    | 29         | Chase21  |
| ...      |            |          |
```

Other useful character functions

`LENGTH(val)` : returns length of a string

```
SELECT LENGTH('cat')
```

```
3
```

`REPLACE(val, m, n)` : replace `m` with `n` in `val`

```
SELECT REPLACE('kayak', 'k', 'y')
```

```
yayay
```

Rounding

`ROUND(column, m)` : round `column` to `m` decimal

```
SELECT Total, ROUND(Total, 1) AS Round1, ROUND(Total, 0) AS Whole
FROM Invoice
```

```
| Total      | Round1 | Whole |
|-----------|--------|-------|
| 11.94      | 11.9   | 12    |
| 14.91      | 14.9   | 15    |
| 0.99       | 1.0    | 1     |
| 5.94       | 5.9    | 6     |
| 7.96       | 8.0    | 8     |
| ...        |        |       |
```

Truncating

`TRUNC(column, m)` : truncates `column` to `m` decimal

```
SELECT DISTINCT Total, ROUND(Total, 1) AS Dec1, TRUNC(Total, 1) AS Trun1
FROM Invoice
```

```
| Total | Dec1 | Trun1 |
|-------|------|-------|
| 15.86 | 15.9 | 15.8  |
| 13.86 | 13.9 | 13.8  |
| 8.94  | 8.9  | 8.9   |
| 1.99  | 2.0  | 1.9   |
| 7.96  | 8.0  | 7.9   |
```

Modulo

`MOD(column1, column2)` : returns remainder of division

```
SELECT MOD(14, 4)
```

```
2
```

Modulo

`MOD(column1, column2)` : returns remainder of division

```
SELECT MOD(14, 2)
```

```
0
```

```
SELECT MOD(15, 2)
```

```
1
```

Modulo

Do we have an even amount of employees?

```
SELECT MOD(COUNT(Employee),1)
FROM Employee
```

```
0
```

Yes.

Let's practice!

INTRODUCTION TO ORACLE SQL

Working with NULL values

INTRODUCTION TO ORACLE SQL

Hadrien Lacroix
Content Developer

What are NULL values?

- No value
- Not the same as 0
- Arithmetic expressions with `NULL` evaluate to `NULL`
  - `NULL + 10 = NULL`
- Aggregate functions usually ignore `NULL` values
  - `COUNT` doesn't count `NULL` values in a columns

Why do we care about null values?

Real world data isn't perfect.

- Clean data
- Analyze missing data

Testing if a value is NULL

`=` can't be used to test for `NULL` values

Instead use:

- `IS NULL`

```
SELECT * FROM Customer WHERE LastName IS NULL
```

- `IS NOT NULL`

```
SELECT * FROM Customer WHERE LastName IS NOT NULL
```

NVL

`NVL(x, y)` : convert `x` , which may contain a null value, to `y` , a non-null value.

```
SELECT NVL(HireDate, '11/19/2004')
FROM Employee
```

NULLIF

`NULLIF(x, y)` : Compares `x` and `y` , returns

- `NULL` if `x` = `y`
- `x` if they are not equal

```sql
SELECT c.CustomerId, i.BillingCity, c.City, NULLIF(i.BillingCity, c.City)
FROM Invoice i, Customer c
```

```
| CustomerId | BillingCity | City      | NULLIF |
|------------|-------------|-----------|--------|
| 48         | Oslo        | Amsterdam | Oslo   |
| 49         | Boston      | Vienne    | Boston |
| 59         | London      | London    | NULL   |
```

COALESCE

`COALESCE` : returns first non-null value in a list

```
SELECT CustomerId, COALESCE(phone, email, fax) AS ContactMethod
FROM Customer
```

```
| CustomerId | ContactMethod            |
|------------|--------------------------|
| 59         | +91 080 22289999         |
| 58         | manoj.pareek@rediff.com  |
| 57         | +56 (0)2 635 4444        |
```

Let's practice!

INTRODUCTION TO ORACLE SQL

Using conversion functions

INTRODUCTION TO ORACLE SQL

Hadrien Lacroix
Content Developer

Data types

Data types define what type of data a column can contain.

	Numeric data	Character data	Date data
`AVG`	x		
`SUM`	x		
`MIN`	x	x	x
`COUNT`	x	x	x

Conversion functions convert a column from one data type to another

Conversion functions

- Data type conversion
  - Implicit data type conversion
  - Explicit data type conversion

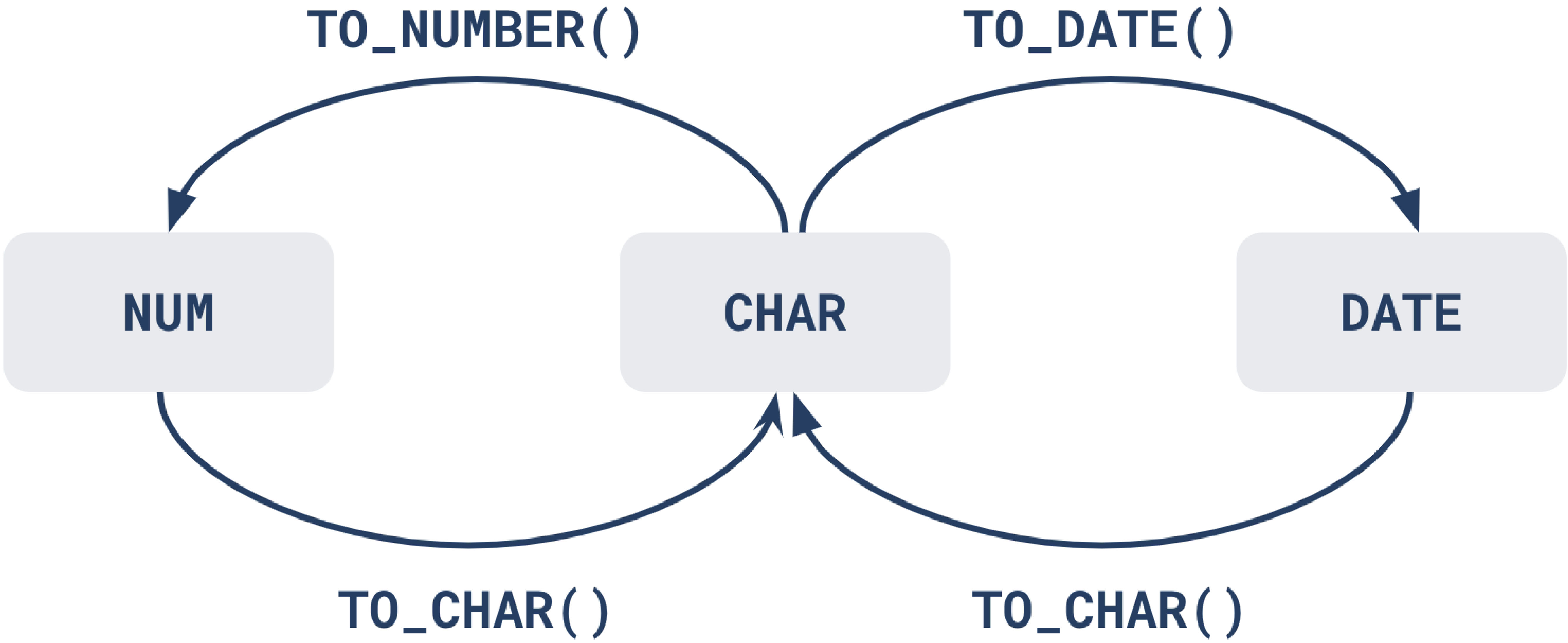
Implicit data type conversion

SQL automatically converts data types

```
SELECT 'Track length: ' || Milliseconds
FROM Track
```

```
| 'Track length: ' || Milliseconds     |
|--------------------------------------|
| Track length: 343719                 |
| Track length: 342562                 |
| ...                                  |
```

Explicit data type conversion



Converting to character data

Convert a **number** string to a character format using the `TO_CHAR` function:

```
SELECT UnitPrice, TO_CHAR(UnitPrice, '$999.99')
FROM InvoiceLine
```

```
| UnitPrice   | TO_CHAR(UnitPrice, '$9.99') |
|-------------|-----------------------------|
| 0.99        | $0.99                       |
| 1.99        | $1.99                       |
| ...         | ...                         |
```

- `$` : Floating dollar sign
- `.` : Decimal position
- `9` : Specifies numeric position. The number of 9's determine the display width
- `0` : Specifies leading zeros
- `,` : Comma position in the number

Converting to character data

Convert a **date** string to a character format using the `TO_CHAR` function:

```
SELECT TO_CHAR(BirthDate, 'DD-MON-YYYY')
FROM Employee
```

```
| TO_CHAR(BirthDate, 'DD-MON-YYYY') |
|-----------------------------------|
| 19-SEP-1947                       |
| ...                               |
```

- `YYYY` : Four digit representation of year
- `YEAR` : Year spelled out
- `MM` : Two digit value of month
- `MONTH` : Full name of month
- `MON` : 3-letter representation of month
- `DY` : 3-letter representation of day of week
- `DAY` : Full name of the day
- `DD` : Numeric day of the month

Converting to numeric data

Convert a character string to a number format using the `TO_NUMBER` function:

```sql
SELECT TO_NUMBER('$15,000.75', '$999,999.99')
FROM DUAL
```

```
| TO_NUMBER('$15,000.75', '$999,999.99') |
|----------------------------------------|
| 15000.75                               |
```

- `$` : Floating dollar sign
- `.` : Decimal position
- `9` : Specifies numeric position. The number of 9's determine the display width
- `0` : Specifies leading zeros
- `,` : Comma position in the number

Converting to date data

Convert a character string to a date format using the `TO_DATE` function:

```
SELECT TO_DATE('2016-01-31','YYYY-MM-DD')
FROM DUAL
```

```
| TO_DATE('2016-01-31','YYYY-MM-DD')     |
|----------------------------------------|
| 31-JAN-16                              |
```

- `YYYY` : Four digit representation of year
- `YEAR` : Year spelled out
- `MM` : Two digit value of month
- `MONTH` : Full name of month
- `MON` : 3-letter representation of month
- `DY` : 3-letter representation of day of week
- `DAY` : Full name of the day
- `DD` : Numeric day of the month

Which data type conversion should you use?

- Always use explicit conversion
  - Easier to read and maintain
  - Code will continue to work

Let's practice!

INTRODUCTION TO ORACLE SQL

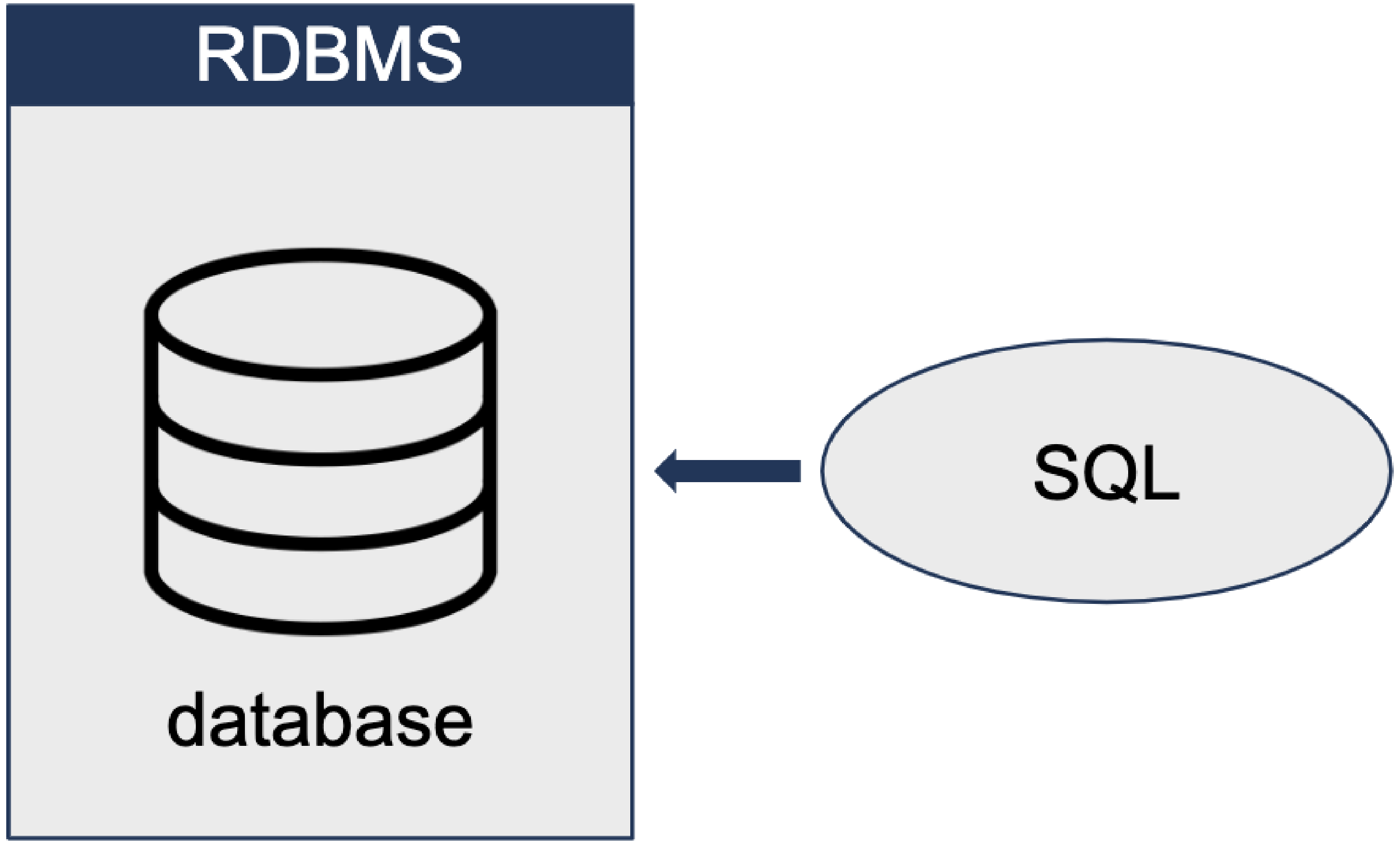
Congratulations!

INTRODUCTION TO ORACLE SQL

Hadrien Lacroix
Content Developer

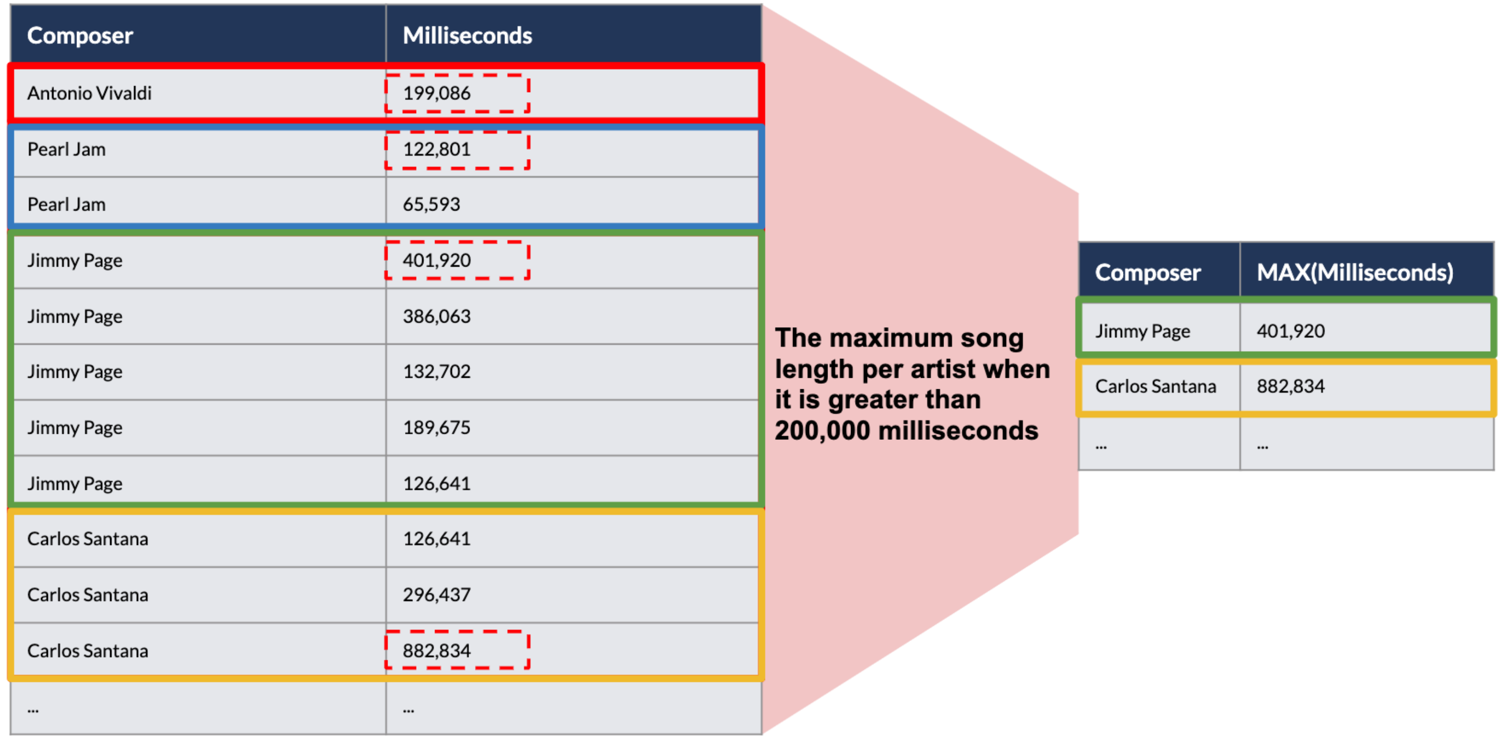
Chapter 1

- Why learn Oracle
- Write first query
- Retrieve data
- Order data
- Restrict data
- Work with strings



Chapter 2

- Group functions
- Data types
- More restricting
- `WHERE` vs `HAVING`



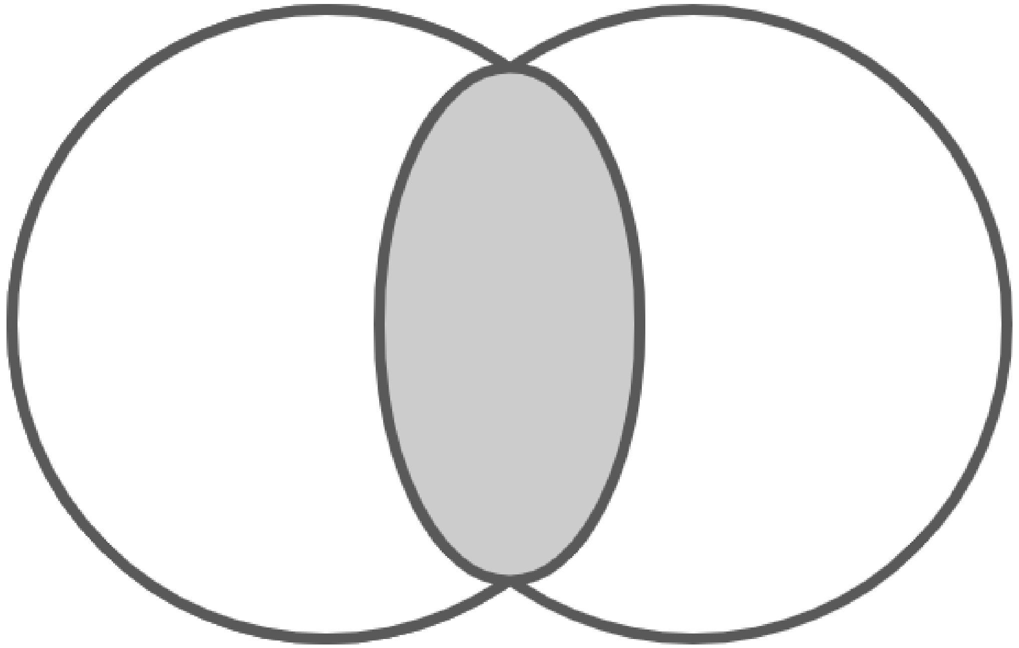
Composer	Milliseconds
Antonio Vivaldi	199,086
Pearl Jam	122,801
Pearl Jam	65,593
Jimmy Page	401,920
Jimmy Page	386,063
Jimmy Page	132,702
Jimmy Page	189,675
Jimmy Page	126,641
Carlos Santana	126,641
Carlos Santana	296,437
Carlos Santana	882,834
...	...

The maximum song length per artist when it is greater than 200,000 milliseconds

Composer	MAX(Milliseconds)
Jimmy Page	401,920
Carlos Santana	882,834
...	...

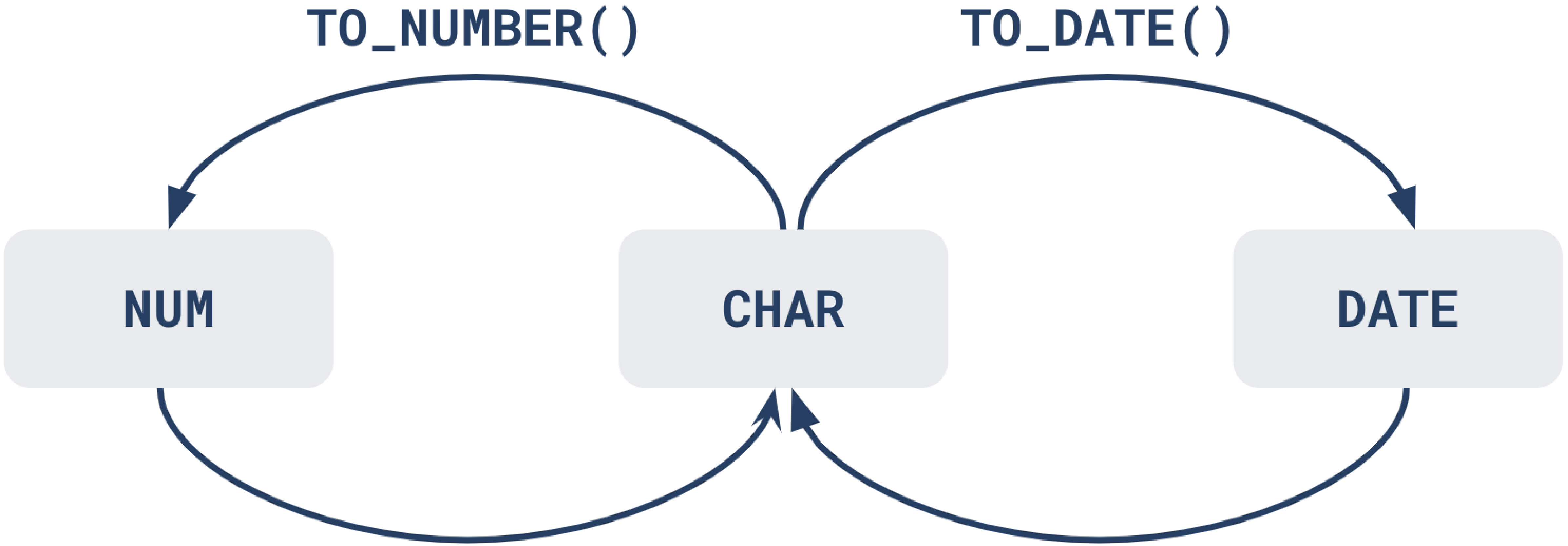
Chapter 3

- `INNER JOIN`
- `OUTER JOIN`
- `SELF JOIN`
- `CROSS JOIN`
- `UNION` , `INTERSECT` , `MINUS`



Chapter 4

- Order of execution
- Customize outputs
- Dealing with missing data



Good bye!

INTRODUCTION TO ORACLE SQL