AGENDA

1. Introduction to **SQL**
2. Retrieving data from **Tables**
3. Filtering **Data**

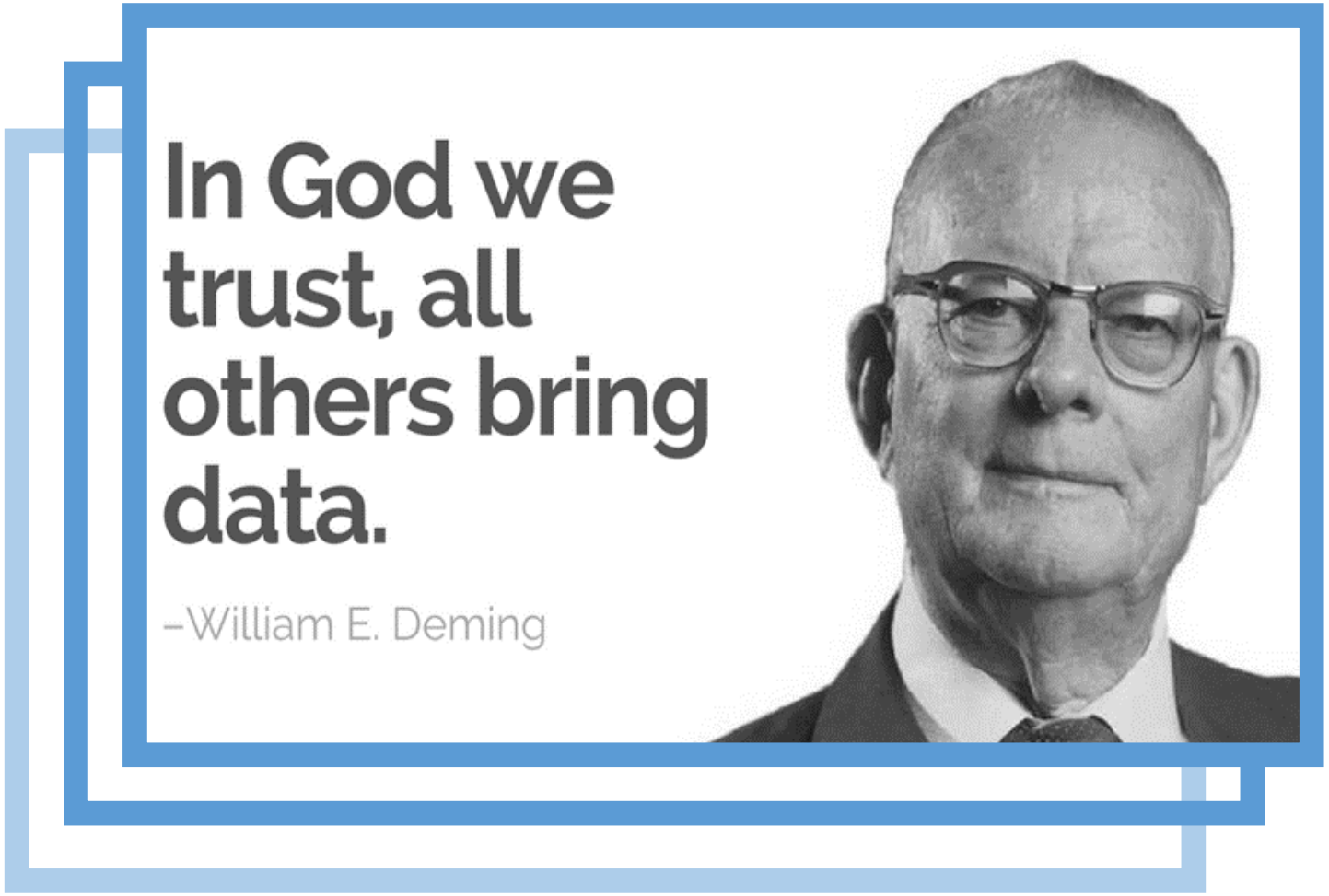
Data Science Academy

I

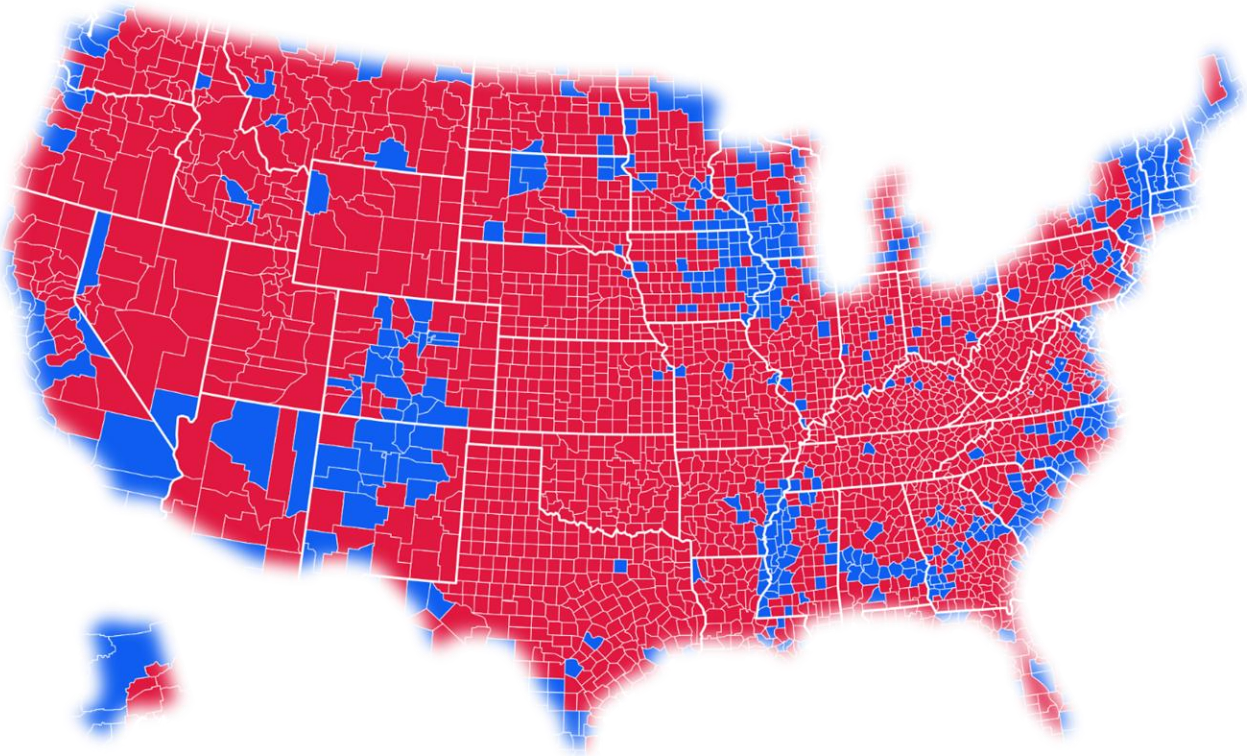
Introduction to SQL

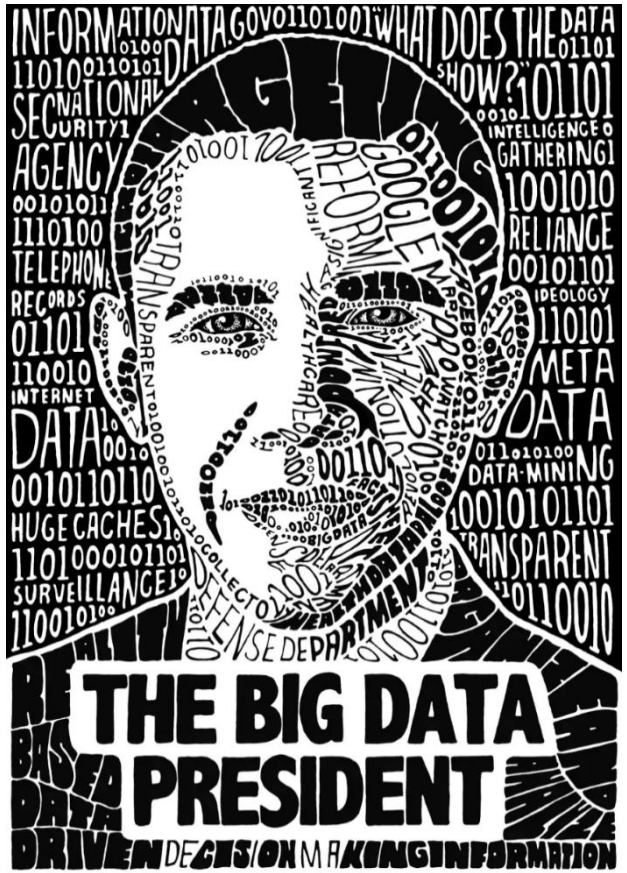
Data Science Academy

THE VALUE OF DATA

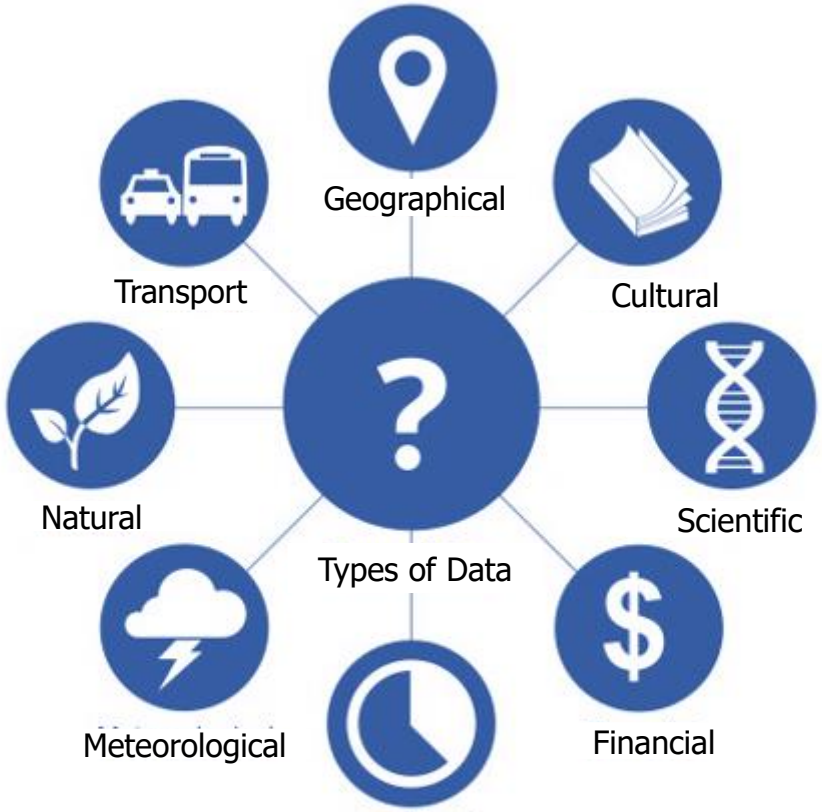


How the Big Data affects elections?



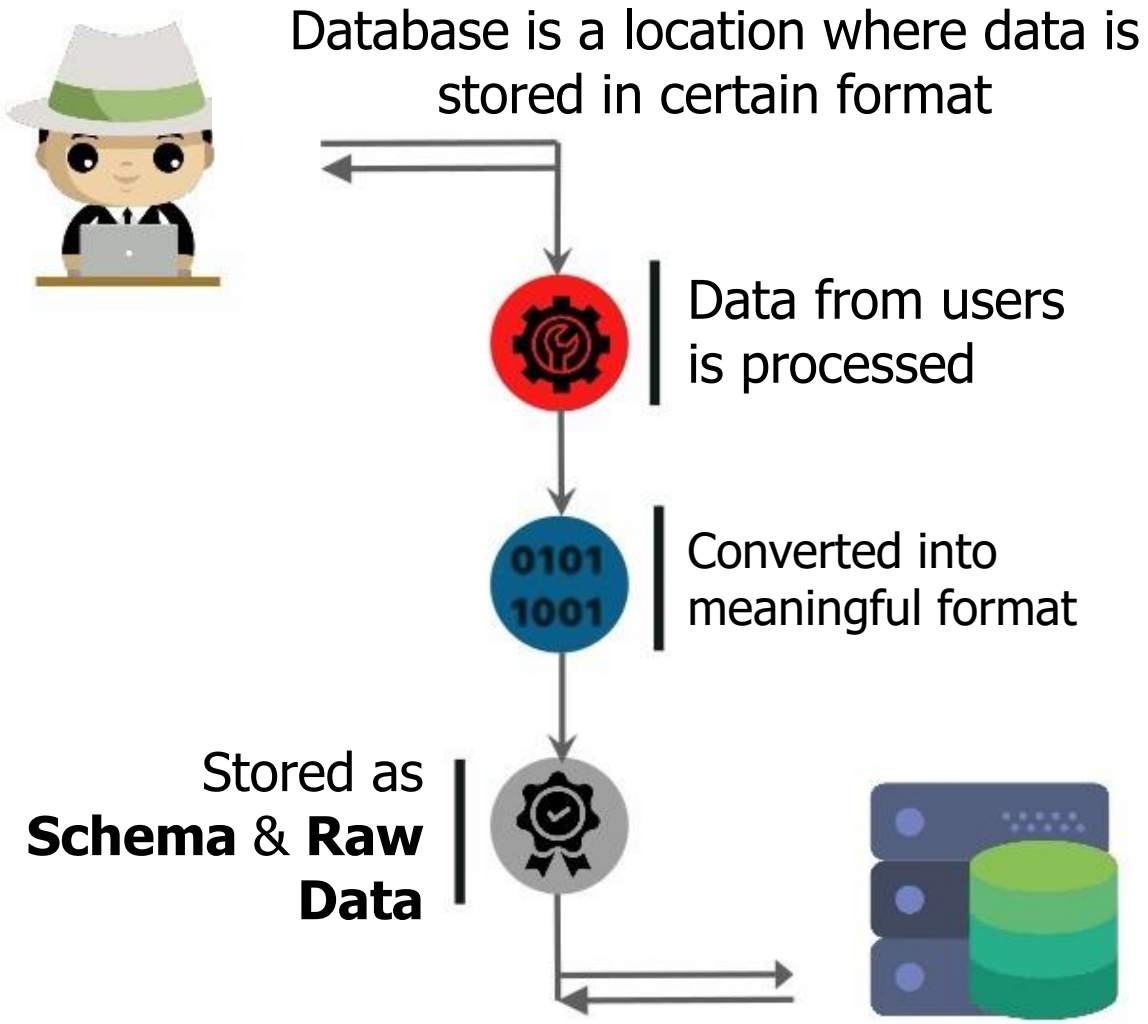


What is Data?



Data is a collection of facts, figures and values from different source

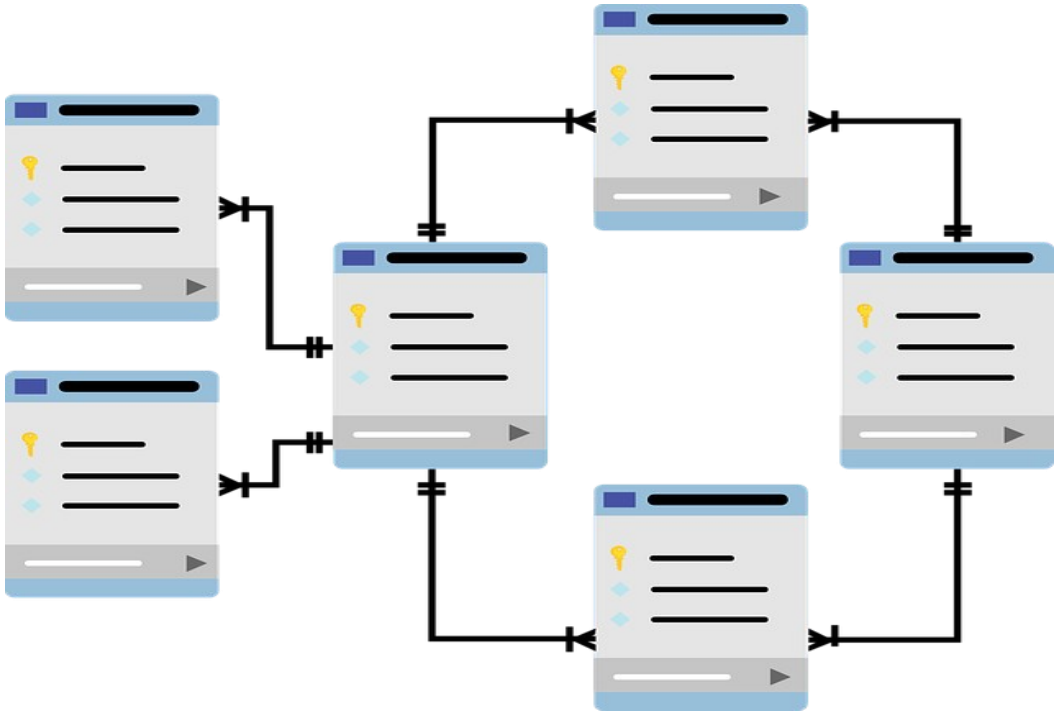
What is Database?



Few Popular Databases

What is Relational Database(RDBMS)?

- RDBMS stores the data into collection of tables which might be realted by common fields(columns).



What is SQL*Plus

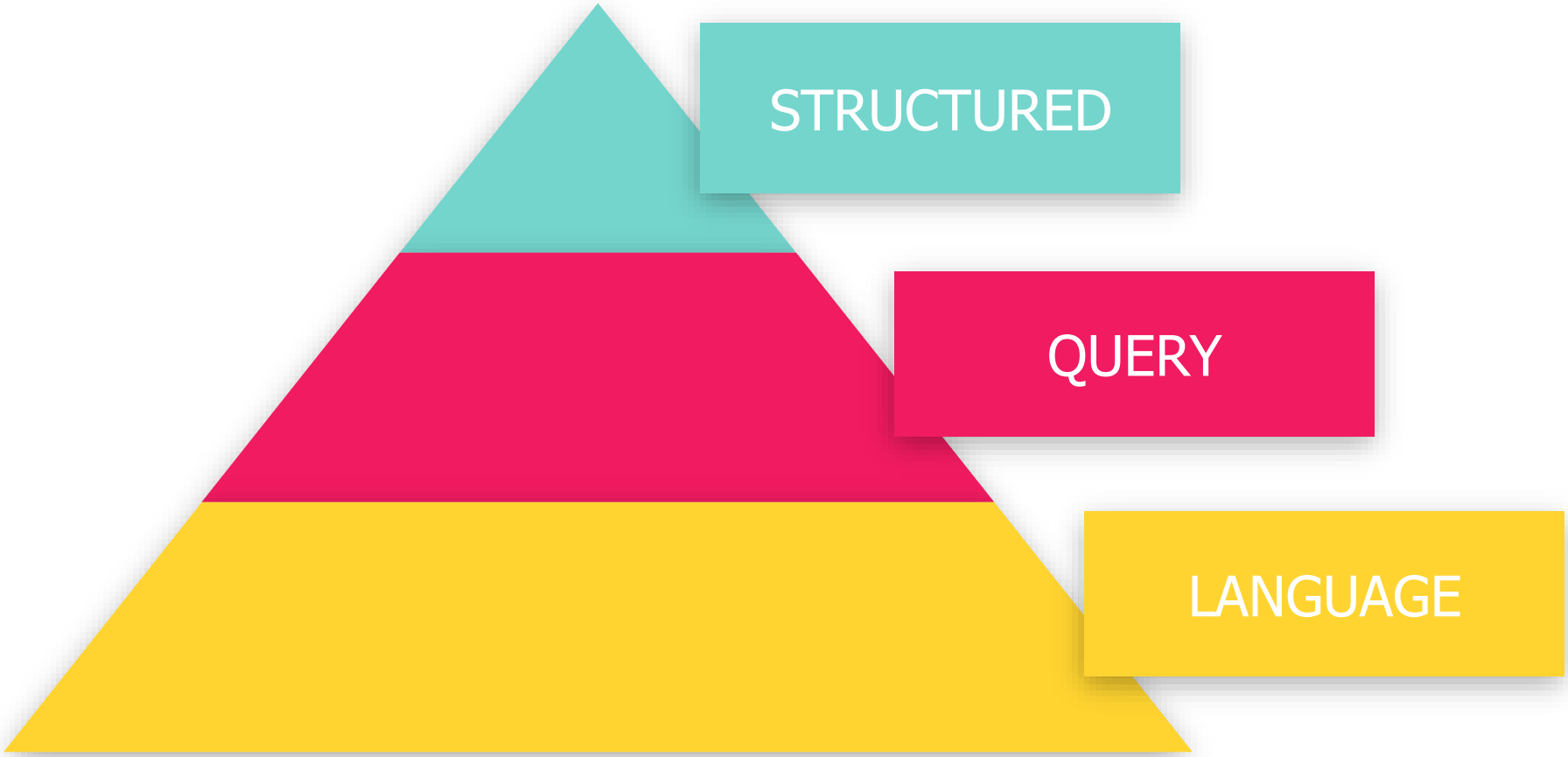
- SQL*Plus is an interactive and batch query tool that is installed with every Oracle Database installation. It has a command-line user interface, a Windows Graphical User Interface (GUI) and the SQL*Plus web-based user interface.

SQL*Plus

SQL*Plus has its own commands and environment, and it provides access to the Oracle Database. It enables you to enter and execute SQL, PL/SQL, SQL*Plus and operating system commands to perform the following:

- Format, perform calculations on, store, and print from query results
- Examine table and object definitions
- Develop and run batch scripts
- Perform database administration

S-Q-L



Features of SQL

SQL has well-defined standards

SQL is very easy to learn

With the help of SQL data can be retrieved from multiple tables with different content

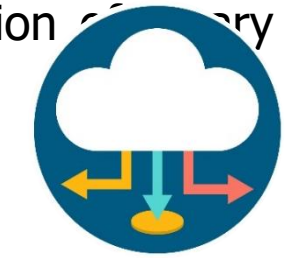
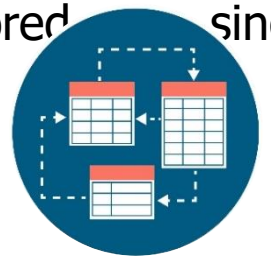
Probability of code in SQL is a prominent feature

Oracle Database

Oracle Database allows you to quickly and safely store and retrieve data.

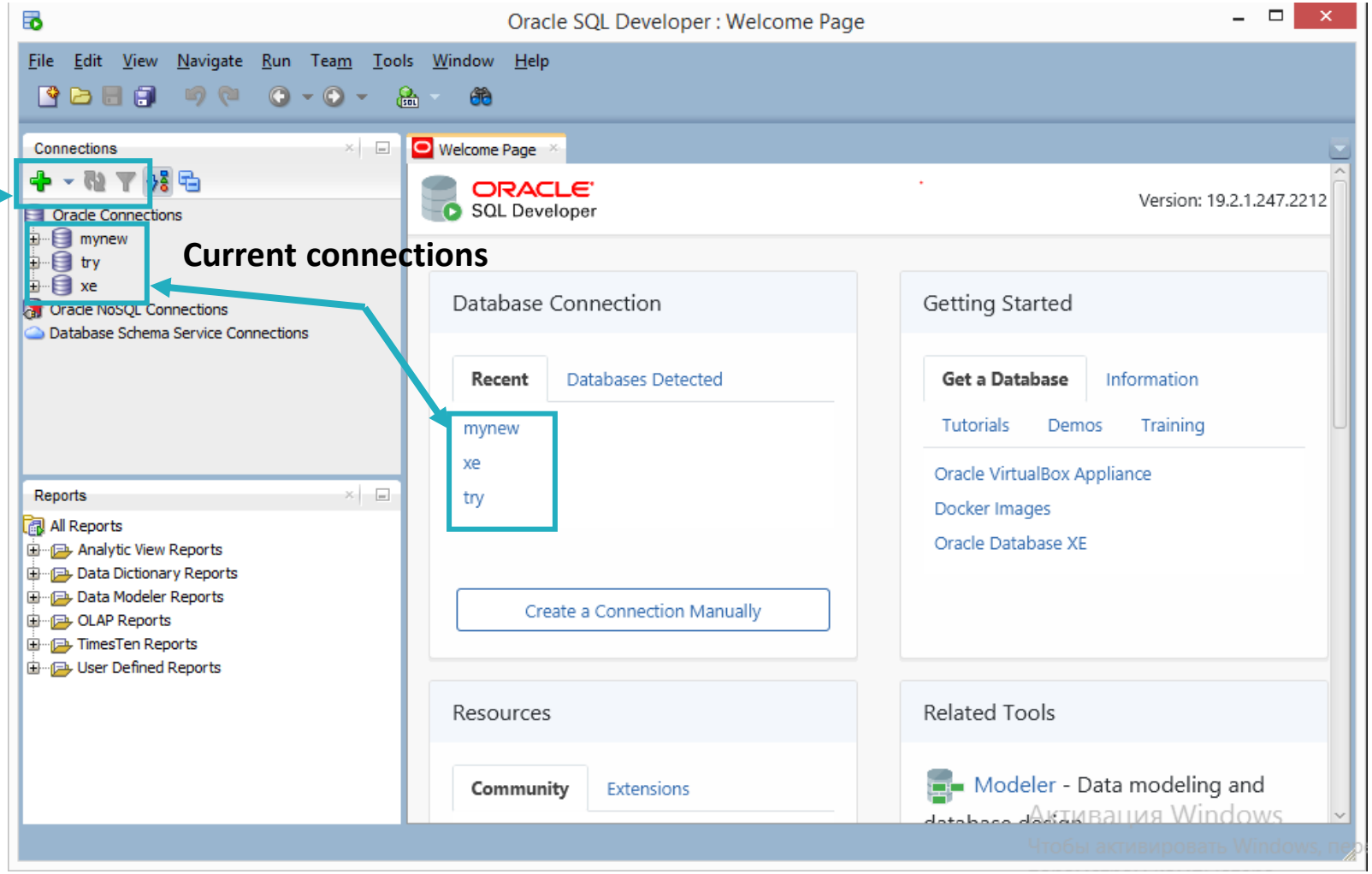
Oracle Database:

- **Cross-platform** – Oracle Database is cross-platform. It can run on various hardware across operating systems including *Windows Server*, *Unix*, and various distributions of *GNU/Linux*.
- **Communication of different platforms** – Oracle Database has its networking stack that allows application from a different platform to communicate with the Oracle Database smoothly. For example, applications running on Windows can connect to the Oracle Database running on Unix.
- **ACID-compliant** – Oracle is ACID-compliant Database that helps maintain data integrity and reliability.
- **Commitment to open technologies** – Oracle is one of the first Database that supported GNU/Linux in the late 1990s before GNU/Linux become a commerce product. It has been supporting this open platform since then
- **CLOB** – CLOB is a data type used by various database management systems, including Oracle and DB2. It stores large amounts of character data, up to 4 GB in size.
- **BLOB** – Blob is a collection of binary data stored as a single entity in a database management system

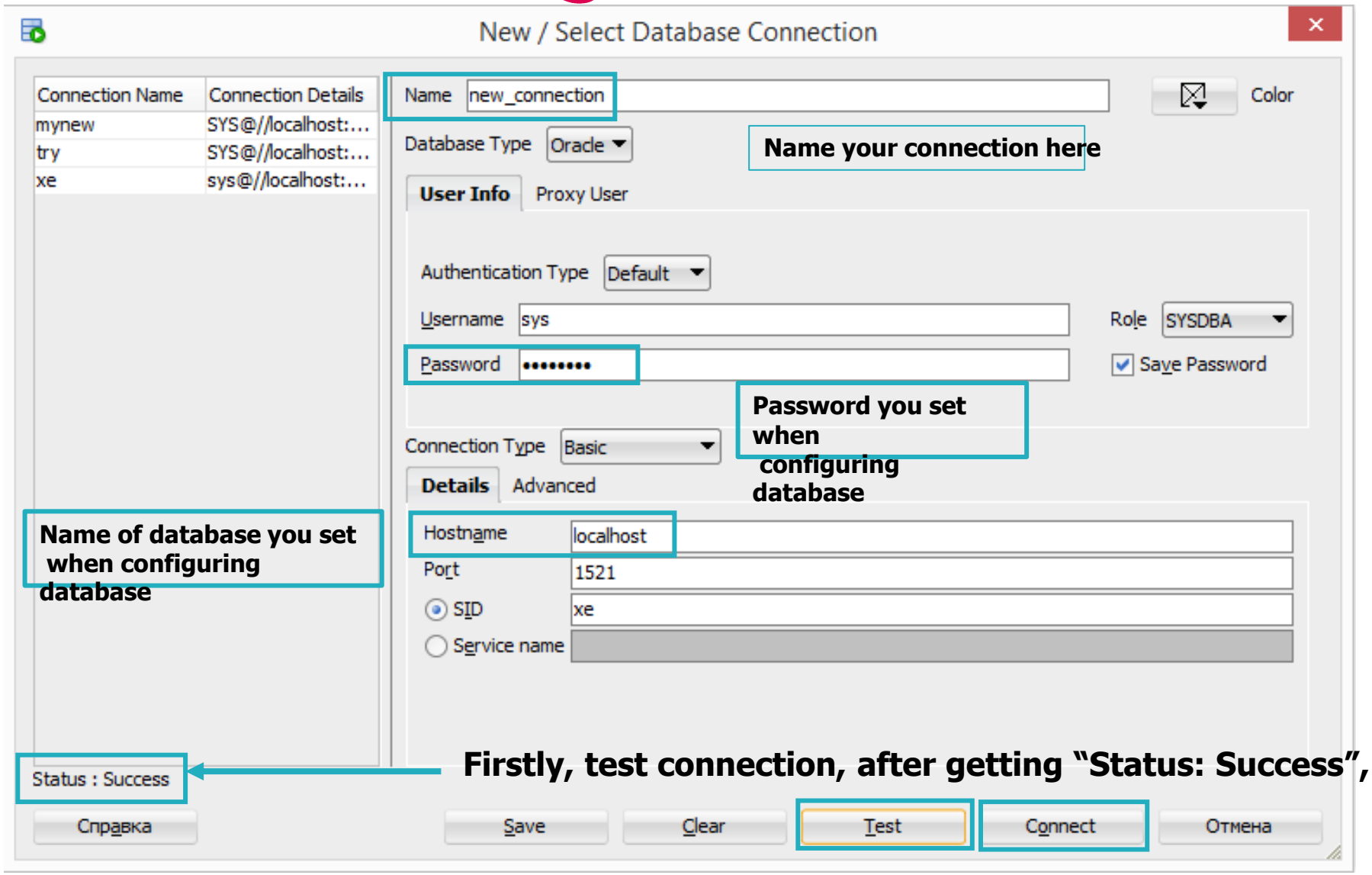





SQL Developer. Welcome Page

- Create new connection



Creating connection



Writing SQL queries

To run whole script

Here we can see name
of a current connection

To run only selected part of query

Welcome Page | new_connection

new_connection

Worksheet | Query Builder

```
CREATE TABLE PERSONS_1(
    P_ID NUMBER NOT NULL,
    P_NAME VARCHAR2(30)
    );
```

Here we write queries

Script Output

Task completed in 0,24 seconds

```
Table PERSONS_1 created.
```

Result of a query will be displayed here

A Database Structure

A database contains schema, which describe the organization of the database.

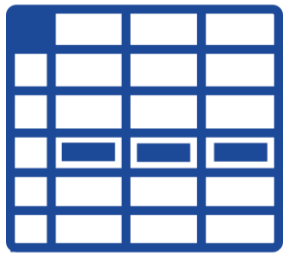
A schema can contain:

- Tables
- Views
- Sequences
- Synonyms
- Indexes
- Saved procedures
- Packages and more.

Schema 1	Schema 2	Schema 3
• Table 1	• Table 1	• Table 1
• Table 2	• Table 2	• Table 2
• Table 3	• Table 3	• Table 3
•	•	•
•	•	•
•	•	•
• Table n	• Table n	• Table n

What is a table?

Table is a collection of data in a tabular form



FIELDS (COLUMNS)

COLUMN NAMES

RECORD (ROW)

ID	FULL_NAME	DEPARTMENT_ID	CITY
1	Kamil Jabbarzade	45	Baku
2	Aida Guliyeva	32	Ganja
3	Aytaj Bayramli	2	Sumgayit
4	Nijat Aliyev	45	Baku

COLUMN VALUES

DATA TYPES-STRING

Types	Description	Size
VARCHAR2(size [BYTE | CHAR])	Variable-length character string.	From 1 byte to 4KB.
NVARCHAR2(size)	Variable-length Unicode character string having maximum length size characters.	Maximum size is determined by the national character set definition, with an upper limit of 4000 bytes. You must specify size for NVARCHAR2.
CHAR [(size [BYTE | CHAR])]	Fixed-length character data of length size bytes or characters.	Maximum size is 2000 bytes or characters. Default and minimum size is 1 byte.
NCHAR[(size)]	Fixed-length character data of length size characters. The number of bytes can be up to two times size for AL16UTF16 encoding and three times size for UTF8 encoding.	Maximum size is determined by the national character set definition, with an upper limit of 2000 bytes. Default and minimum size is 1 character.
CLOB	CLOB data type stores variable-length character data (character large object) in the database character set that can be single-byte or multi-byte	Supports more than 4 GB
BLOB	Like other binary types, BLOB strings are not associated with a code page. In addition, BLOB strings do not hold character data.	A BLOB (binary large object) is a varying-length binary string that can be up to 2,147,483,647 characters long.

DATA TYPES-DATE/TIME

Types	Description	Size
DATE	Valid date range : From January 1, 4712 BC, to December 31, 9999 AD. The default format is determined explicitly by the NLS_DATE_FORMAT parameter or implicitly by the NLS_TERRITORY parameter.	The size is fixed at 7 bytes.
TIMESTAMP [(fractional_seconds_precision)]	This data type contains the datetime fields YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND. It contains fractional seconds but does not have a time zone.	The size is 7 or 11 bytes, depending on the precision.
TIMESTAMP [(fractional_seconds_precision)] WITH TIME ZONE	This data type contains the datetime fields YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, TIMEZONE_HOUR, and TIMEZONE_MINUTE. It has fractional seconds and an explicit time zone.	The size is fixed at 13 bytes.

SQL Commands

DDL

Data Definition Language

DML

Data Manipulation Language

DCL

Data Control Language

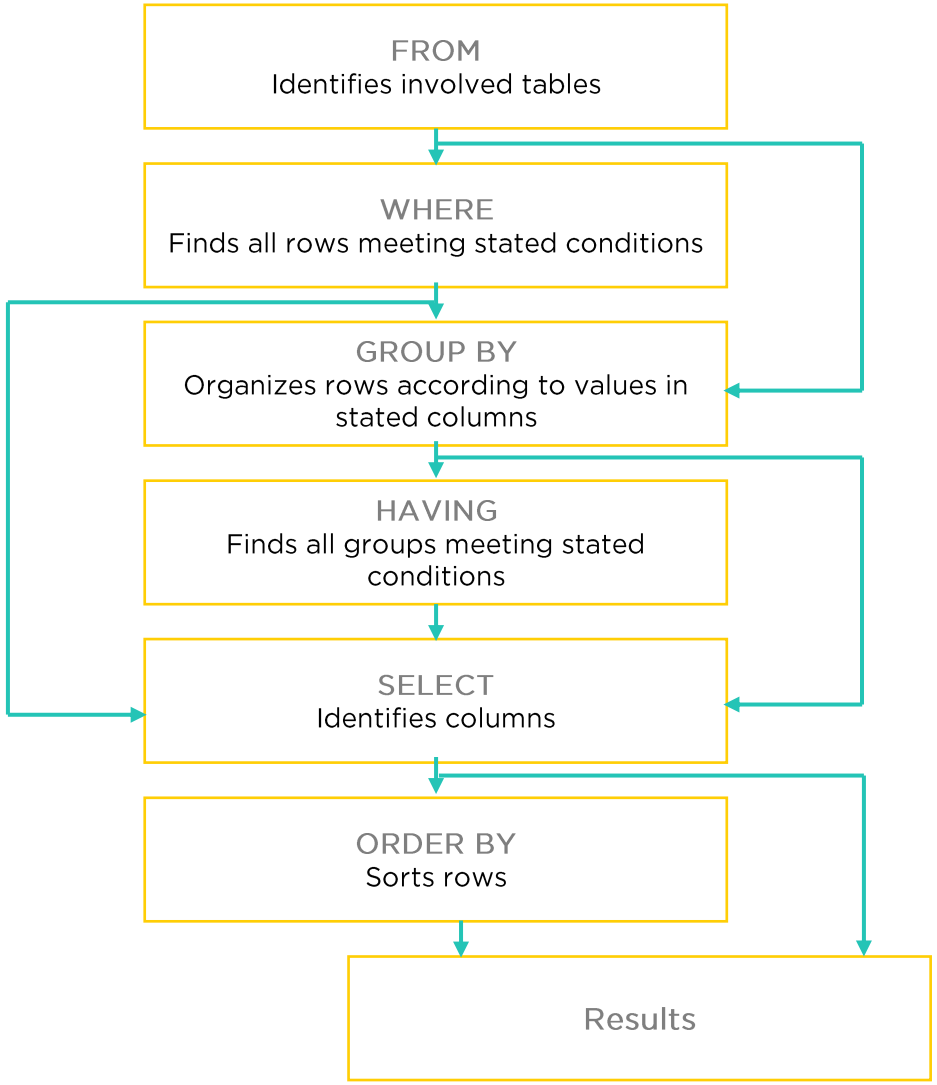
TCL

Transaction Control Language

Operator	Description
CREATE	is used to create new database objects (table, procedure, view and etc.)
DROP	is used to delete objects from database
ALTER	is used to change the structure of the objects
TRUNCATE	command deletes the data inside a table, but not the table itself
SELECT	is used to retrieve data from database
INSERT	is used to insert new records in a table
UPDATE	is used to modify the existing data in a table
DELETE	is used to delete existing data in a table
GRANT	is used to provide any user access privileges or other privileges for the database
REVOKE	is used to take back permissions from any user.
COMMIT	commits a Transaction
ROLLBACK	rollbacks a transaction in case of any error occurs
SAVEPOINT	sets a savepoint within a transaction
SET TRANSACTION	specify characteristics for the transaction

SQL STATEMENT PROCESSING ORDER

- Given example shows the order of processing of commands in SQL statement (based on van der Lans, 2006 p.100)



II
Retrieving data from Tables

Data Science Academy

GETTING DATA FROM DATABASE

Select Operator

- The Oracle SELECT statement is used to retrieve records from one or more tables in an Oracle database.

-

- The data returned is stored in a result table, called the result-set.

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

SELECT
  column_1,
  column_2,
  ...
FROM
  table_name;
```

Qualifying SELECT

- To retrieve data from one or more columns of a table, you use the SELECT statement with the following syntax:

```
SELECT
  column_1,
  column_2,
  ...
FROM
  table_name;
```

In this SELECT statement:

- First, specify the table name from which you want to query the data.
- Second, indicate the columns from which you want to return the data. If you have more than one column, you need to separate each by a comma (,).

Oracle SELECT examples

- To query data from multiple columns, you specify a list of comma-separated column names.

- The following example shows how to query data from the customer_id, name, and credit_limit columns of the customer table.

```
SELECT
   customer_id,
   name,
   credit_limit
FROM
   customers;
```

- The following picture illustrates the result:

CUSTOMER_ID	NAME	CREDIT_LIMIT
35	Kimberly-Clark	400
36	Hartford Financial Services Group	400
38	Kraft Heinz	500
40	Fluor	500
41	AECOM	500
44	Jabil Circuit	500
45	CenturyLink	500
47	General Mills	600
48	Southern	600
50	Thermo Fisher Scientific	700

MATHEMATICAL OPERATORS

Operator	Description
=	Equal
<>	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	If you know the exact value you want to return for at least one of the columns

ORDER BY

- In Oracle, a table stores its rows in unspecified order regardless of the order which rows were inserted into the database. To sort data, you add the ORDER BY clause to the SELECT statement as follows:

```
SELECT
    column_1,
    column_2,
    column_3,
    ...
FROM
    table_name
ORDER BY
    column_1 [ASC | DESC] [NULLS FIRST | NULLS LAST],
    column_1 [ASC | DESC] [NULLS FIRST | NULLS LAST],
    ...
```

ORDER BY

- To sort the result set by a column, you list that column after the **ORDER BY** clause. Following the column name is a sort order that can be:
  - `ASC` for sorting in ascending order
  - `DESC` for sorting in descending order

Therefore, the following expression:

```
ORDER BY name ASC
```

is equivalent to the following:

```
ORDER BY name
```

Oracle ORDER BY clause example

- To sort the customer data by names alphabetically in ascending order, you use the following statement:

```
SELECT
    name,
    address,
    credit_limit
FROM
    customers
ORDER BY
    name ASC;
```

NAME	ADDRESS	CREDIT_LIMIT
3M	Via Frenzy 6903, Roma,	1200
ADP	Langstr 14, Zuerich, ZH	700
AECOM	2102 E Kimberly Rd, Davenport, IA	500
AES	33 Fulton St, Poughkeepsie, NY	1200
AIG	12817 Coastal Hwy, Ocean City, MD	2400
AT&T	55 Church Hill Rd, Reading, PA	1200
AbbVie	6445 Bay Harbor Ln, Indianapolis, IN	100
Abbott Laboratories	3310 Dixie Ct, Saginaw, MI	200
Advance Auto Parts	2674 Collingwood St, Detroit, MI	3700
Aetna	200 E Fort Ave, Baltimore, MD	2400

Oracle ORDER BY clause example

- To sort customer by name alphabetically in descending order, you explicitly use DESC after the column name in the ORDER BY clause as follows:

```
SELECT
    name,
    address,
    credit_limit
FROM
    customers
ORDER BY
    name DESC;
```

- The following picture shows the result that customers sorted by names alphabetically in descending order:

NAME	ADDRESS	CREDIT_LIMIT
eBay	Via Del Disegno 194, Milano,	1500
Yum Brands	Ruella Delle Spiriti, Roma,	500
Xerox	9936 Dexter Ave, Detroit, MI	400
Xcel Energy	1540 Stripes Crt, Baden-Daettwil, AG	400
World Fuel Services	Theresienstr 15, Munich,	2400
Whole Foods Market	4200 Yosemite Ave S, Minneapolis, MN	1200
Whirlpool	18305 Van Dyke St, Detroit, MI	200
Western Refining	5565 Baynton St, Philadelphia, PA	2400
Western Digital	33 Pine St, Lockport, NY	1200
WestRock	Chrottenweg, Bern, BE	5000

Sorting rows by multiple columns example

- To sort multiple columns, you separate each column in the ORDER BY clause by a comma

- For example, to sort contacts by their first names in ascending order and their last names in descending order, you use the following stateme

```
SELECT
first_name,
last_name
FROM
contacts
ORDER BY
first_name,
last_name DESC;
```

Cristine	Bell
Daina	Combs
Daniel	Glass
Daniel	Costner
Darron	Robertson
Debra	Herring
Dell	Wilkinson
Delpha	Golden
Deneen	Hays
Denny	Daniel
Diane	Higgins
Dianne	Sen
Dianne	Derek

Sorting rows by column's position example

- You don't need to specify the column names for sorting the data. If you prefer you can use the positions of the column in the ORDER BY clause.

See the following statement:

- In this example the position of name column is 1 and credit limit column is 2.
- In the ORDER by clause, we used these column positions to instruct the Oracle to sort the rows.

```
SELECT
    name,
    credit_limit
FROM
    customers
ORDER BY
    2 DESC,
    1;
```

Sorting by date example

- This example uses the ORDER BY clause to sort orders by order date:

```sql
SELECT
    order_id,
    customer_id,
    status,
    order_date
FROM
    orders
ORDER BY
    order_date DESC;
```

ORDER_ID	CUSTOMER_ID	STATUS	ORDER_DATE
88	6	Shipped	01-NOV-17
94	1	Shipped	27-OCT-17
1	4	Pending	15-OCT-17
14	48	Shipped	28-SEP-17
15	49	Shipped	27-SEP-17
17	17	Shipped	27-SEP-17
36	51	Shipped	05-SEP-17
57	68	Shipped	24-AUG-17
28	6	Canceled	15-AUG-17

Eliminating duplicates

- The DISTINCT clause is used in a SELECT statement to filter duplicate rows in the result set. It ensures that rows returned are unique for the column or columns specified in the SELECT clause.

```
SELECT   DISTINCT category
FROM     Product
```

Category
Gadgets
Photography
Household

Compare to:

```
SELECT   category
FROM     Product
```

Category
Gadgets
Gadgets
Photography
Household

Oracle column alias

- When you query data from a table, Oracle uses the column names of the table for displaying the column heading. However, sometimes, the column names are quite uncertain for describing the meaning of data. To better describe the data displayed in the output, you can substitute a column alias for the column name in the query results.

- For instance, instead of using first name and last name, you might want to use forename and surname for display names of employees:

```
SELECT
  first_name AS forename,
  last_name  AS surname
FROM
  employees;
```

FORENAME	SURNAME
Summer	Payne
Rose	Stephens
Annabelle	Dunn
Tommy	Bailey
Blake	Cooper
Jude	Rivera
Tyler	Ramirez
Ryan	Gray
Elliot	Brooks
Elliott	James

Oracle column alias

- The `AS` keyword is used to distinguish between the column name and the column alias. Because the `AS` keyword is optional, you can skip it as follows:

```
SELECT
  first_name forename,
  last_name surname
FROM
  employees;
```

Forename	Surname
Summer	Payne
Rose	Stephens
Annabelle	Dunn
Tommy	Bailey
Blake	Cooper
Jude	Rivera
Tyler	Ramirez
Ryan	Gray
Elliot	Brooks

- By default, Oracle capitalizes the column heading in the query result. If you want to change the letter case of the column heading, you need to enclose it in quotation marks ("")

```
SELECT
  first_name "Forename",
  last_name "Surname"
FROM
  employees;
```

Using Oracle column alias for expression

- Besides making the column headings more meaningful, you can use the column alias for an expression, for example, instead of :

```
SELECT
 first_name || ' ' || last_name
FROM
 employees;
```

can be written :

```
SELECT
 first_name || ' ' || last_name AS "Full Name"
FROM
 employees;
```

Oracle table alias

- Similar to a column name, you can assign a table name an alias. A table alias is a temporary name for a table in a query. You specify a table alias after the table name either with or without the `AS` keyword:

```
table_name AS table_alias
table_name table_alias
```

- Without the table alias, you qualify a column by using the following form:

```
table_name.column_name
```

- However, you must use an alias instead of the table name after you assign the table a table alias:

```
table_alias.column_name
```

Oracle table alias

- A table alias improves the readability of the query and reduces the number of keystrokes:

```
SELECT
  e.first_name employee,
  m.first_name manager
FROM
  employees e
INNER JOIN employees m
ON
  m.employee_id = e.employee_id;
```

EMPLOYEE	MANAGER
Summer	Rose
Jaxon	Tommy
Liam	Tommy
Jackson	Tommy
Callum	Tommy
Ronnie	Tommy
Mia	Tommy
Ava	Tommy
Ella	Tommy

- In this example, the `employees` table joins to itself. This technique is called self-join .Because a table only can appear in a query once, you must use the table alias to give the employees two different names, `e` for employees and `m` for managers

III
Filtering Data

Data Science Academy

Let’s Filter The Data

01

Use of Where clause for filtering numeric value

- SELECT * FROM sales WHERE total_amount > 1000;
- SELECT * FROM sales WHERE total_amount != 44;
- SELECT * FROM sales WHERE total_amount^44;
- SELECT * FROM sales WHERE quantity <= 10;

02

Use of Where clause for filtering text value

- SELECT * FROM sales WHERE sales_date = ’09-feb-2015’;
- SELECT * FROM product WHERE color = ‘RED’;

03

Use of Where clause for comparing column values

- SELECT * FROM sales WHERE total_amount > sales_amount;

Introduction to Oracle WHERE clause

- The `WHERE` clause specifies a search condition for rows returned by the `SELECT` statement. The following illustrates the syntax of the `WHERE` clause:

```
SELECT
   column_1,
   column_2,
   ...
FROM
   table_name
WHERE
   search_condition
ORDER BY
   column_1,
   column_2;
```

- The `WHERE` clause appears after the `FROM` clause but before the `ORDER BY` clause. Following the `WHERE` keyword is the `search_condition` that defines a condition which returned rows must satisfy

Operators in WHERE/HAVING Clauses

Operator	Meaning
=	Equals
<>	Not equal
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
in (...)	Contained in a set of items
like	Wildcard match
(...)	Order of Operations
and	True if left side and right side are true
or	True if left side or right side are true
not	Negates condition

Oracle WHERE example

- The following example returns only products whose names are `'Kingston'`:

```
SELECT
   product_name,
   description,
   list_price,
   category_id
FROM
   products
WHERE
   product_name = 'Kingston';
```

- In this example, Oracle evaluates the clauses in the following order: `FROM` `WHERE` and `SELECT`
- First, the `FROM` clause specified the table for quering data.
- Second, the `WHERE` clause filtered rows based on the condition e.g., `product_name = 'Kingston'`).
- Third, the `SELECT` clause chose the columns that should be returned.

BETWEEN

- The BETWEEN operator allows you to specify a range to test. When you use the BETWEEN operator to form a search condition for rows returned by a SELECT statement, only rows whose values are in the specified range are returned.
- The following illustrates the syntax of the BETWEEN operator:

```
expression [ NOT ] BETWEEN low AND high
```

- The following statement returns products whose standard costs are between 500 and 600:

```
SELECT
    product_name,
    standard_cost
FROM
    products
WHERE
    standard_cost BETWEEN 500 AND 600
ORDER BY
    standard_cost;
```

Oracle BETWEEN dates example

- The following statement returns the orders placed by customers between December 1, 2016, and December 31, 2016:

```sql
SELECT
    order_id,
    customer_id,
    status,
    order_date
FROM
    orders
WHERE
    order_date BETWEEN DATE '2016-12-01'
AND DATE '2016-12-31'
ORDER BY
    order_date;
```

ORDER_ID	CUSTOMER_ID	STATUS	ORDER_DATE
87	7	Canceled	01-DEC-16
85	4	Pending	01-DEC-16
83	16	Shipped	02-DEC-16
82	44	Shipped	03-DEC-16
81	49	Shipped	13-DEC-16
80	3	Shipped	13-DEC-16
79	2	Shipped	14-DEC-16
102	45	Shipped	20-DEC-16

Oracle NOT BETWEEN dates example

- To query products whose standard costs are not between 500 and 600, you add the NOT operator to the above query as follows:

```sql
SELECT
    product_name,
    standard_cost
FROM
    products
WHERE
    standard_cost NOT BETWEEN 500 AND 600
ORDER BY
    product_name;
```

PRODUCT_NAME	STANDARD_COST
ADATA ASU800SS-128GT-C	37.78
ADATA ASU800SS-512GT-C	113.29
AMD 100-5056062	1343.84
AMD 100-505989	2128.67
AMD 100-506061	706.99
AMD FirePro S7000	936.42
AMD FirePro W9100	2483.38
AMD Opteron 6378	651.92
ASRock C2750D4I	339.55

Introduction to Oracle AND operator

- The `AND` operator is a logical operator that combines Boolean expressions and returns true if both expression are true. If one of the expressions is false, the `AND` operator returns false.
- The syntax of the `AND` operator is as follows:

```
expression_1 AND expression_2
```

- The following table illustrates the result when you combine the true, false, and a NULL value using the `AND` operator.

	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

Oracle AND operator example

- The following example finds orders of the customer 2 with the pending status:

```
SELECT
order_id,
customer_id,
status,
order_date
FROM
orders
WHERE
status = 'Pending'
AND customer_id = 2
ORDER BY
order_date;
```

Here is the result:

ORDER_ID	CUSTOMER_ID	STATUS	ORDER_DATE
78	2	Pending	14-DEC-15
44	2	Pending	20-FEB-17

Introduction to Oracle OR operator

- The `OR` operator is a logical operator that combines Boolean expressions and returns true if one of the expressions is true.
- The following illustrates the syntax of the `OR` operator:

```
expression_1 OR expression_2
```

- The following table shows the results the `OR` operator between true, false, and a NULL value.

	TRUE	**FALSE**	**NULL**
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

Oracle OR operator example

- The following example finds orders whose status is pending or canceled:

```sql
SELECT
    order_id,
    customer_id,
    status,
    order_date
FROM
    orders
WHERE
    status = 'Pending'
    OR status = 'Canceled'
ORDER BY
    order_date DESC;
```

ORDER_ID	CUSTOMER_ID	STATUS	ORDER_DATE
1	4	Pending	15-OCT-17
28	6	Canceled	15-AUG-17
31	46	Canceled	12-AUG-17
21	21	Pending	27-MAY-17
5	5	Canceled	09-APR-17
69	44	Canceled	17-MAR-17
70	45	Canceled	21-FEB-17
44	2	Pending	20-FEB-17
46	58	Pending	20-FEB-17
10	44	Pending	24-JAN-17

AND vs OR

- For example, the following query finds order placed by customer id 44 and has status canceled or pending:

```sql
SELECT
   order_id,
   customer_id,
   status,
   salesman_id,
   order_date
FROM
   orders
WHERE
   (
      status = 'Canceled'
      OR status = 'Pending'
   )
   AND customer_id = 44
ORDER BY
   order_date;
```

ORDER_ID	CUSTOMER_ID	STATUS	SALESMAN_ID	ORDER_DATE
10	44	Pending	(null)	24-JAN-17
69	44	Canceled	54	17-MAR-17

IN operator

- The Oracle IN operator determines whether a value matches any values in a list or a subquery.

- The syntax of Oracle IN operator that determines whether an expression matches a list of value is as follows:

```
expression [NOT] IN (v1,v2,...)
```

- …and syntax of an expression matches a subquery:

```
expression [NOT] IN (subquery)
```

Oracle IN operator example

- The following statement finds all orders which are in charge of the salesman id 54, 55, and 56:

```sql
SELECT
    order_id,
    customer_id,
    status,
    salesman_id
FROM
    orders
WHERE
    salesman_id IN (
        54,
        55,
        56
    )
ORDER BY
    order_id;
```

ORDER_ID	CUSTOMER_ID	STATUS	SALESMAN_ID
1	4	Pending	56
5	5	Canceled	56
44	2	Pending	55
49	61	Shipped	55
50	62	Pending	55
54	65	Shipped	56
56	67	Canceled	55
61	2	Shipped	54
69	44	Canceled	54
71	46	Shipped	54

Oracle IN operator example

- Similarly, the following example retrieves sales orders whose statuses are Pending or Canceled:

```sql
SELECT
    order_id,
    customer_id,
    status,
    salesman_id
FROM
    orders
WHERE
    status IN(
        'Pending',
        'Canceled'
    )
ORDER BY
    order_id;
```

ORDER_ID	CUSTOMER_ID	STATUS	SALESMAN_ID
1	4	Pending	56
5	5	Canceled	56
10	44	Pending	(null)
16	16	Pending	(null)
21	21	Pending	(null)
22	22	Canceled	(null)
27	43	Canceled	(null)
28	6	Canceled	57
31	46	Canceled	(null)

Oracle NOT IN operator example

- The example shows how to find orders whose statuses are not Shipped and Canceled:

```
SELECT
    order_id,
    customer_id,
    status,
    salesman_id
FROM
    orders
WHERE
    status NOT IN(
        'Shipped',
        'Canceled'
    )
ORDER BY
    order_id;
```

ORDER_ID	CUSTOMER_ID	STATUS	SALESMAN_ID
1	4	Pending	56
10	44	Pending	(null)
16	16	Pending	(null)
21	21	Pending	(null)
44	2	Pending	55
46	58	Pending	62
50	62	Pending	55
55	66	Pending	59
68	9	Pending	(null)

Oracle IN vs OR

- The following example shows how to get the sales orders of salesman 60, 61, and 62:

```sql
SELECT
    customer_id,
    status,
    salesman_id
FROM
    orders
WHERE
    salesman_id IN(
        60,
        61,
        62
    )
ORDER BY
    customer_id;
```

```sql
SELECT
    customer_id,
    status,
    salesman_id
FROM
    orders
WHERE
    salesman_id = 60
    OR salesman_id = 61
    OR salesman_id = 62
ORDER BY
    customer_id;
```

Note that the expression:

```sql
salesman_id NOT IN (60,61,62);
```

has the same effect as:

```sql
salesman_id != 60   AND salesman_id != 61  AND salesman_id != 62;
```

LIKE with examples

Sometimes, you want to query data based on a specified pattern. For example, you may want to find contacts whose last names start with 'St' or first names end with 'er'. In this case, you use the Oracle LIKE operator.

The syntax of the Oracle LIKE operator is as follows:

```
expresion [NOT] LIKE pattern [ ESCAPE escape_characters ]
```

Expression

The expression is a column name or an expression that you want to test against the pattern.

Pattern

The pattern is a string to search for in the expression. The pattern includes the following wildcard characters:

% (percent) matches any string of zero or more character.

_ (underscore) matches any single character.

Escape_character

The escape_character is a character that appears in front of a wildcard character to specify that the wildcard should not be interpreted as a wildcard but a regular character.

LIKE with examples

% wildcard character examples

The following example uses the % wildcard to find the phones of contacts whose last names start with 'St':

```sql
SELECT
    first_name,
    last_name,
    phone
FROM
    contacts
WHERE
    last_name LIKE 'St%'
ORDER BY
    last_name;
```

The following picture illustrates the result:

FIRST_NAME	LAST_NAME	PHONE
Josie	Steele	+41 69 012 3581
Bill	Stein	+39 6 012 4501
Birgit	Stephenson	+1 608 123 4374
Herman	Stokes	+39 49 012 4777
Violeta	Stokes	+1 810 123 4212
Gonzalo	Stone	+1 301 123 4814
Flor	Stone	+1 317 123 4104

LIKE with examples

- To find the phone numbers of contacts whose last names end with the string 'er', you use the following statement:

```
SELECT
   first_name,
   last_name,
   phone
FROM
   contacts
WHERE
   last_name LIKE '%er'
ORDER BY
   last_name;
```

FIRST_NAME	LAST_NAME	PHONE
Shamika	Bauer	+91 11 012 4853
Stephaine	Booker	+39 55 012 4559
Charlene	Booker	+41 61 012 3537
Annice	Boyer	+1 518 123 4618
Shelia	Brewer	+49 89 012 4129
Annabelle	Butler	+91 80 012 3737
Nichol	Carter	+91 11 012 4813
Barbie	Carter	+41 5 012 3573
Sharee	Carver	+1 215 123 4738
Agustina	Conner	+1 612 123 4399
Daniel	Costner	+1 812 123 4153

NOT LIKE examples

- The NOT operator, if specified, negates the result of the LIKE operator.
- The following example uses the NOT LIKE operator to find contacts whose phone numbers do not start with '+1':

```sql
SELECT
  first_name, last_name, phone
FROM
  contacts
WHERE
  phone NOT LIKE '+1%'
ORDER BY
  first_name;
```

FIRST_NAME	LAST_NAME	PHONE
Adah	Myers	+41 3 012 3553
Adam	Jacobs	+91 80 012 3699
Adrienne	Lang	+39 2 012 4771
Aleshia	Reese	+41 4 012 3563
Alessandra	Estrada	+41 56 012 3527
Amber	Brady	+91 80 012 3837
Annabelle	Butler	+91 80 012 3737
Annelle	Lawrence	+39 10 012 4379
Arlette	Thornton	+91 80 012 3719

_wildcard character examples

- The following example finds the phone numbers and emails of contacts whose first names have the following pattern 'Je_i':

```sql
SELECT
    first_name,
    last_name,
    email,
    phone
FROM
    contacts
WHERE
    first_name LIKE 'Je_i'
ORDER BY
    first_name;
```

FIRST_NAME	LAST_NAME	EMAIL	PHONE
Jeni	Levy	jeni.levy@centene.com	+1 812 123 4129
Jeri	Randall	jeri.randall@nike.com	+49 90 012 4131

Mixed Wildcard characters example

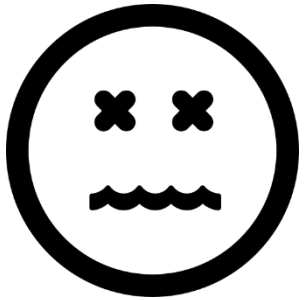
- You can mix the wildcard characters in a pattern. For example, the following statement finds contacts whose first names start with Je followed by two characters and then any number of characters. In other words, it will match any last name that starts with Je and has at least 3 characters:

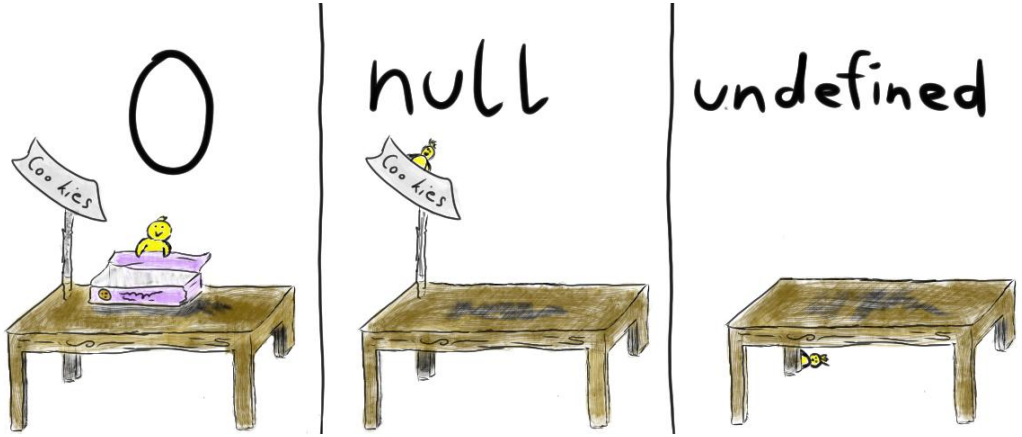
```sql
SELECT
    first_name,
    last_name,
    email,
    phone
FROM
    contacts
WHERE
    first_name LIKE 'Je_%';
```

FIRST_NAME	LAST_NAME	EMAIL	PHONE
Jeannie	Poole	jeannie.poole@aboutmcdonalds.com	+91 80 012 4637
Jeni	Levy	jeni.levy@centene.com	+1 812 123 4129
Jeri	Randall	jeri.randall@nike.com	+49 90 012 4131
Jerica	Brooks	jerica.brooks@northropgrumman.com	+91 11 012 4811
Jermaine	Cote	jermaine.cote@wfscorp.com	+49 91 012 4133
Jess	Nguyen	jess.nguyen@searsholdings.com	+39 2 012 4773
Jessika	Merritt	jessika.merritt@bnymellon.com	+1 612 123 4397

NULLS in SQL

- Whenever we don't have a value, we can put a NULL
- Can mean many things:
  - Value does not exists
  - Value exists but is unknown
  - Value not applicable
  - Etc.
- The schema specifies for each attribute if can be null (*nullable* attribute) or not
- How does SQL cope with tables that have NULLs ?





Introduction to the Oracle IS NULL operator

- In the database world, NULL is special. It is a marker for missing information or the information is not applicable.

```
SELECT * FROM orders
WHERE salesman_id = NULL
ORDER BY order_date DESC;
```

- The following SELECT statement attempts to return all sales orders which do not have a responsible salesman:

- It returns an empty row.

- The query uses the comparison operator (=) to compare the values from the salesman id column with NULL, which is not correct.

Introduction to the Oracle IS NULL example

- The following query returns all sales orders that do not have a responsible salesman:

```
SELECT * FROM orders
WHERE salesman_id IS NULL
ORDER BY order_date DESC;
```

- Here is the partial output of the query:

ORDER_ID	CUSTOMER_ID	STATUS	SALESMAN_ID	ORDER_DATE
14	48	Shipped	(null)	28-SEP-17
15	49	Shipped	(null)	27-SEP-17
17	17	Shipped	(null)	27-SEP-17
36	51	Shipped	(null)	05-SEP-17
29	44	Shipped	(null)	14-AUG-17
31	46	Canceled	(null)	12-AUG-17
30	45	Shipped	(null)	12-AUG-17
20	20	Shipped	(null)	27-MAY-17
21	21	Pending	(null)	27-MAY-17
3	5	Shipped	(null)	26-APR-17

Introduction to the Oracle IS NULL example

```
SELECT * FROM orders
WHERE salesman_id IS NULL
ORDER BY order_date DESC;
```

Here is the partial output of the query:

ORDER_ID	CUSTOMER_ID	STATUS	SALESMAN_ID	ORDER_DATE
14	48	Shipped	(null)	28-SEP-17
15	49	Shipped	(null)	27-SEP-17
17	17	Shipped	(null)	27-SEP-17
36	51	Shipped	(null)	05-SEP-17
29	44	Shipped	(null)	14-AUG-17
31	46	Canceled	(null)	12-AUG-17
30	45	Shipped	(null)	12-AUG-17
20	20	Shipped	(null)	27-MAY-17
21	21	Pending	(null)	27-MAY-17
3	5	Shipped	(null)	26-APR-17

Introduction to the Oracle IS NOT NULL example

To negate the IS NULL operator, you use the IS NOT NULL operator as follows:

```
expression | column IS NOT NULL
```

The operator IS NOT NULL returns true if the expression or value in the column is not null. Otherwise, it returns false.

For example, the following example returns all sales orders which have a responsible salesman:

```
SELECT * FROM orders
WHERE salesman_id IS NOT NULL
ORDER BY order_date DESC;
```

This picture illustrates the partial output:

ORDER_ID	CUSTOMER_ID	STATUS	SALESMAN_ID	ORDER_DATE
88	6	Shipped	61	01-NOV-17
94	1	Shipped	62	27-OCT-17
1	4	Pending	56	15-OCT-17
57	68	Shipped	57	24-AUG-17
28	6	Canceled	57	15-AUG-17
60	1	Shipped	62	30-JUN-17
40	55	Shipped	62	11-MAY-17
41	9	Shipped	59	11-MAY-17
5	5	Canceled	56	09-APR-17
98	48	Shipped	55	18-MAR-17
69	44	Canceled	54	17-MAR-17
70	45	Canceled	61	21-FEB-17
71	46	Shipped	54	21-FEB-17

Thank you!

Data Science Academy