



# ANALYTICS AND DATA SUMMIT 2020

All Analytics. All Data.  
No Nonsense.

February 25-27, 2020

*Hands-on Lab*

## **Property Graph from scratch: data sources to graphs**

by  
Gianni Ceresa   
*Managing Director, DATAlysis*



## Table of Contents

Lab 1: Design your graph .....	32
Lab 2: Create and populate a graph .....	46
Lab 3: Load graph in PGX and query it.....	78





Slide 1




DATAlysis

*Hands on Lab*

# Property Graph from scratch:

## Data sources to graphs



 @G\_Ceresa     [www.dataalysis.ch](http://www.dataalysis.ch)     [info@dataalysis.ch](mailto:info@dataalysis.ch)



## Slide 2







## Slide 3

**Agenda**

	[min]
Introduction to the Hands-on Lab	5
Introduction to Property Graphs and PGX	10
<i>Test connection to Lab environments</i>	5
Introduction to the Labs scenario	5
<i>Lab 1: Design your graph</i>	5
Graph storage, structure, example	15
<i>Lab 2: Create and populate a graph</i>	10
Break	5
PGX: loading graph, PGQL queries, algorithms	10
<i>Lab 3: Load graph in PGX and query it</i>	10
Graph Visualization, Cytoscape, custom viz, REST, PGQL to SQL transformation, Oracle Cloud Always Free Autonomous database, Machine Learning	15
Next steps and conclusion	

@G\_Ceresa



## Slide 4 (and 5)

**Gianni Ceresa**

Managing Director of DATAlysis GmbH (Switzerland)

Working with Business Analytics, EPM tools and "data" for more than 10 years

Oracle ACE Director 

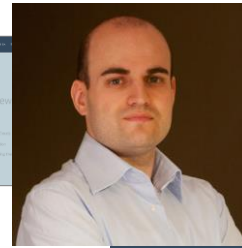
Part-time blogger on [gianniceresa.com](http://gianniceresa.com)

Full-time IRC (freenode | #obihackers) resident

Same group on Telegram <http://telegram.me/obihackers>

ODC (ex OTN) forums addict

Technology geek (or just geek in general)



**DATAlysis**

 @G\_Ceresa



Slide 6

**450+ Technical Experts  
Helping Peers Globally**

**ORACLE®**  
ACE PROGRAM

 **ORACLE®**  
ACE Director

 **ORACLE®**  
ACE

 **ORACLE®**  
ACE Associate

[bit.ly/OracleACEProgram](https://bit.ly/OracleACEProgram)

Nominate yourself or someone you know: [acenomination.oracle.com](https://acenomination.oracle.com)

 @G\_Ceresa



Have a look at the Oracle ACE Program, the various levels, the directory where to search current ACEs and feel free to nominate somebody you believe deserves it or even yourself.



Slide 7

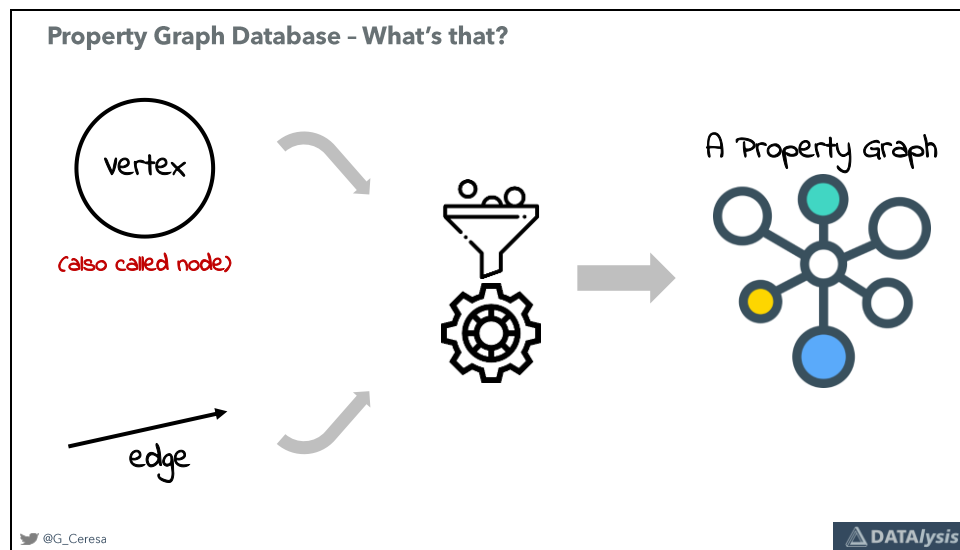
## Introducing Property Graphs and PGX

 @G\_Ceresa

 DATAlysis



Slide 8

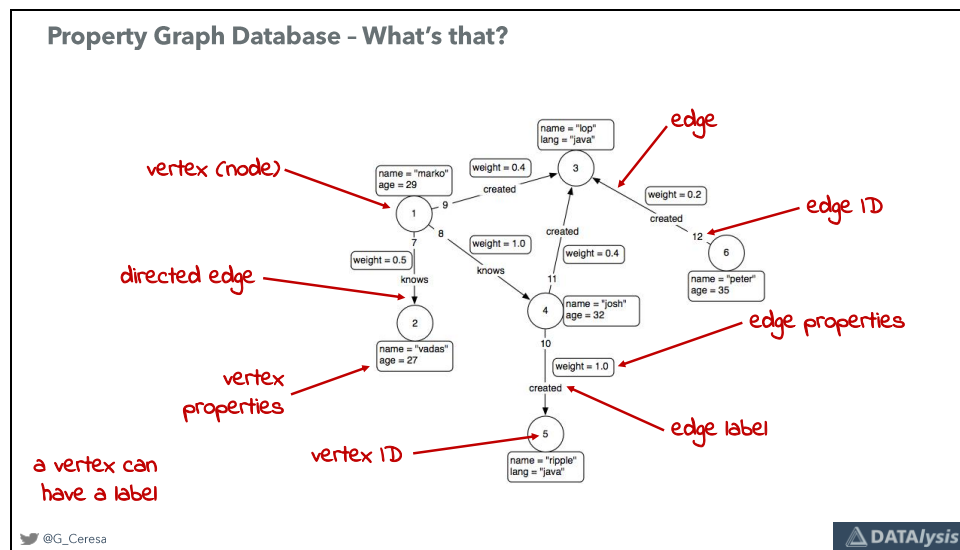


A Property Graph is composed by only 2 kind of objects: vertices (nodes) and edges.

Take some vertices, take some edges, mix all that together and done: you have a graph.



## Slide 9



This is a visualization of an example graph: the 2 components, nodes and edges, are clearly visible.

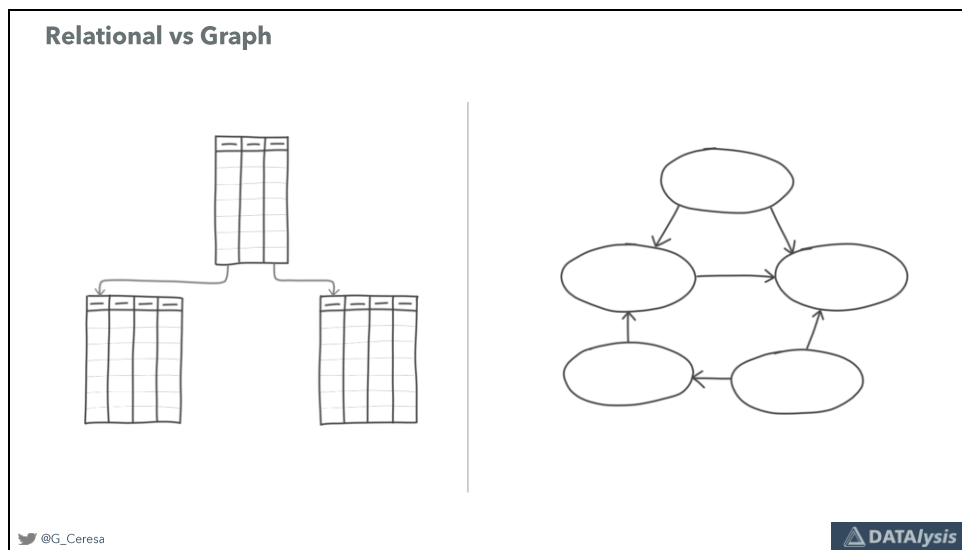
Both have an ID to uniquely identify each element. Nodes can have a label, it isn't mandatory, and in Oracle they can also be more than one. A label is used to qualify the kind of node. For example it could be "customer" or "product" or whatever else.

Edges often have a label, but it isn't mandatory, and it's generally a verb to define the relationship connecting the 2 nodes.

Both nodes and edges can have properties. They are simply key-values entries and they are all independent one from the other. One node can have 10 properties, another 200, another none.



Slide 10

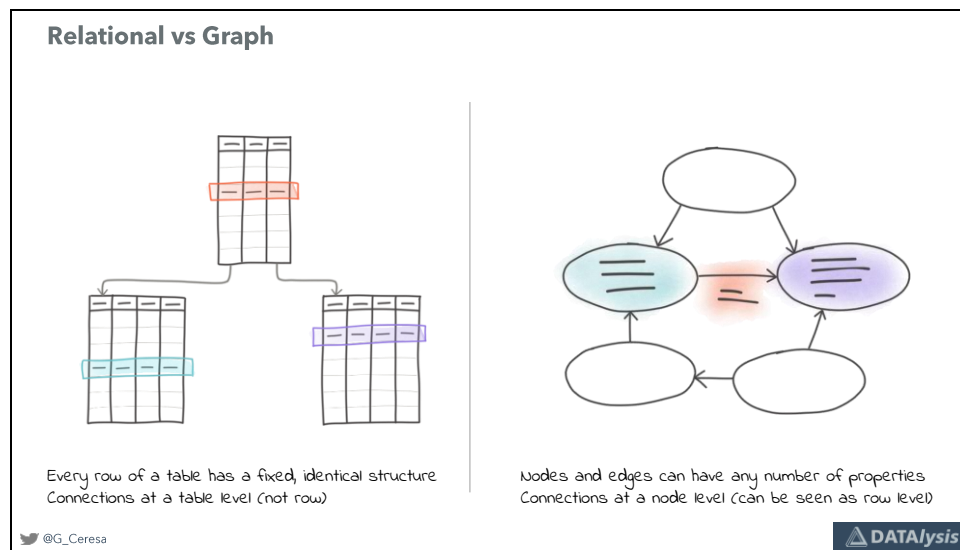


Some tables with joins, primary key – foreign key, between them.  
Tables have a given set of columns.

The graph is some nodes and some edges.



Slide 11



Every single row in those tables always has the exact same set of rows (except if you have a big blob or varchar column and store XML or Json inside, but let's not start talking dirty early in the morning).

The relation isn't at a row level but defined at the table level and all the rows must follow it.

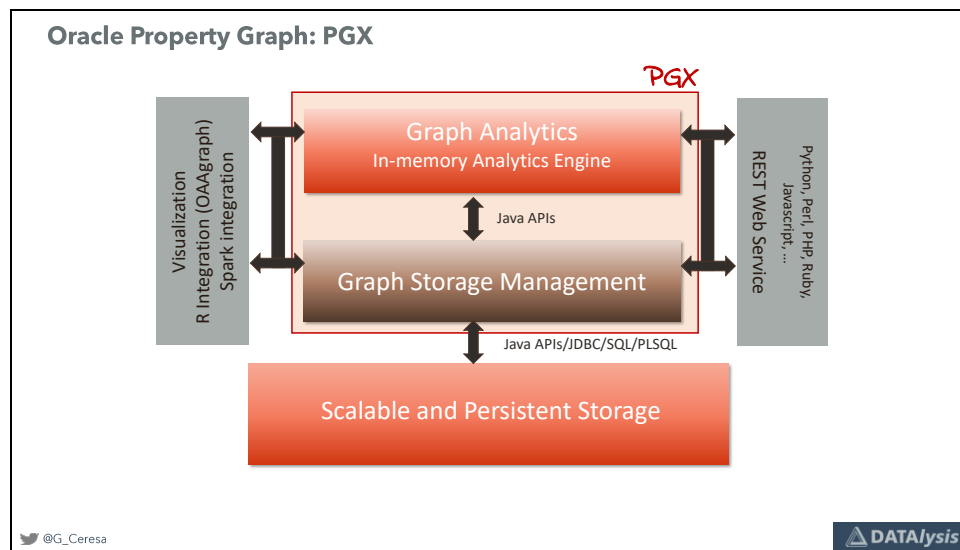
In the graph every node has a different list of properties: one can have 3, one 4, no mandatory structure.

What is a table in the relational database can be an edge in a graph: a mapping table can be turned into an edge with the attributes to qualify it as properties (for example a customer buying products).





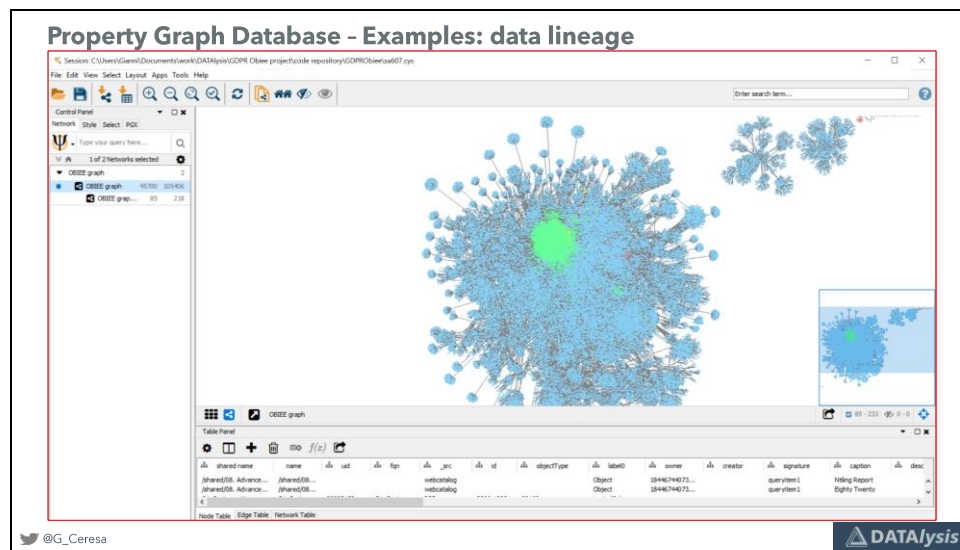
Slide 12



In Oracle the graph “engine”, the brain, is called PGX. It’s the acronym for Parallel Graph Analytics. PGX doesn’t implement a full storage layer, it instead use existing options which could be an Oracle Database, an Oracle NoSQL database or HBase. Even a simple file in the filesystem. PGX “speaks” REST: this allow to connect to it easily from almost any kind of tool or language.



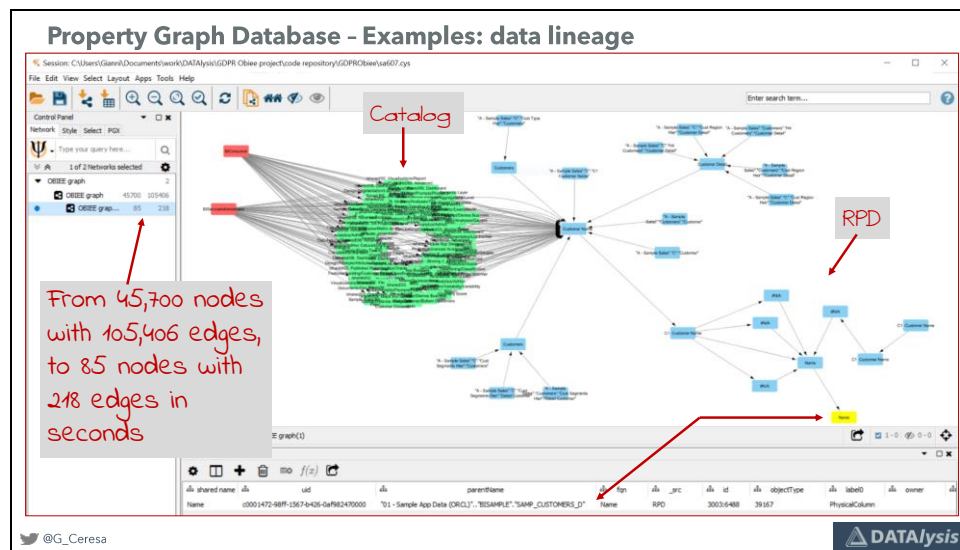
Slide 13



Examples of what a property graph can be used for is data lineage. Imagine having an analytical platform like Oracle Analytics Cloud or Oracle Analytic Server (or the good old OBIEE). Some ETL processes and data source. All these elements have many mappings and connections between them. They can perfectly represented by a graph.



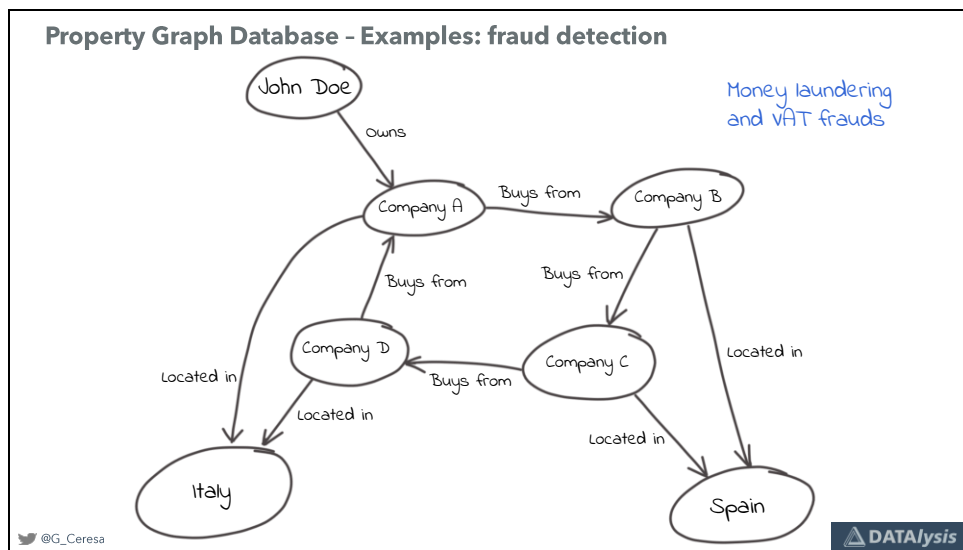
Slide 14



Thanks to the graph it's possible to easily perform impact analysis: what are all the elements I will have to fix if I decide to remove a given column from a physical source in my Oracle Analytics Cloud? The graph provides an answer in less than a second thanks to the in-memory processing.



Slide 15



Another usage of graphs can be fraud detection: finding loops of money circulating and coming back at the origin.



Slide 16

**Property Graph Database - Examples: shortest path**

Examples of graphs and graphs analytics can be seen when traveling from a location A to a location B :

*Finding shortest path between 2 nodes of a graph*



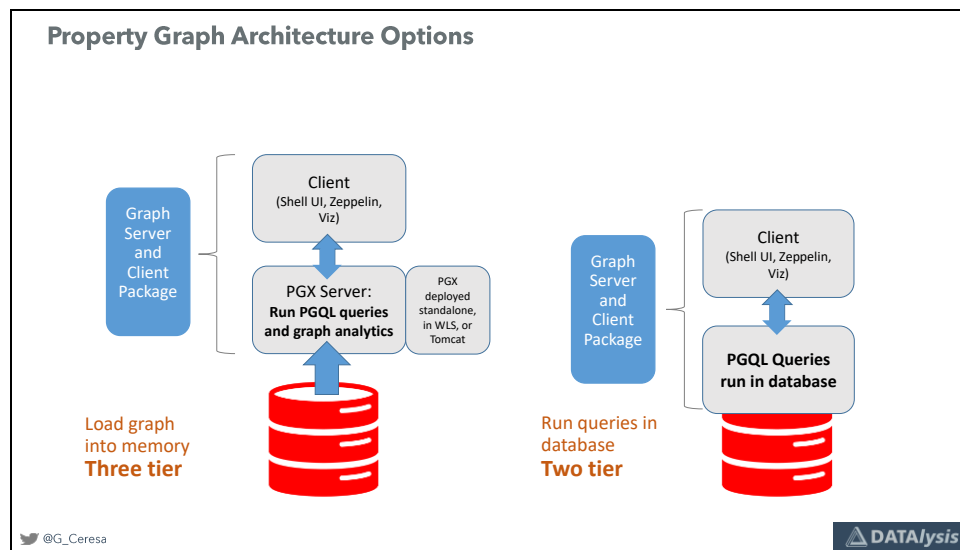
@G\_Ceresa

DATAlysis

One of the simplest and most obvious usages is finding paths. Imagine at all the possible flights existing. How do you go from A to B in the fastest way? In the cheapest one? Etc.



Slide 17

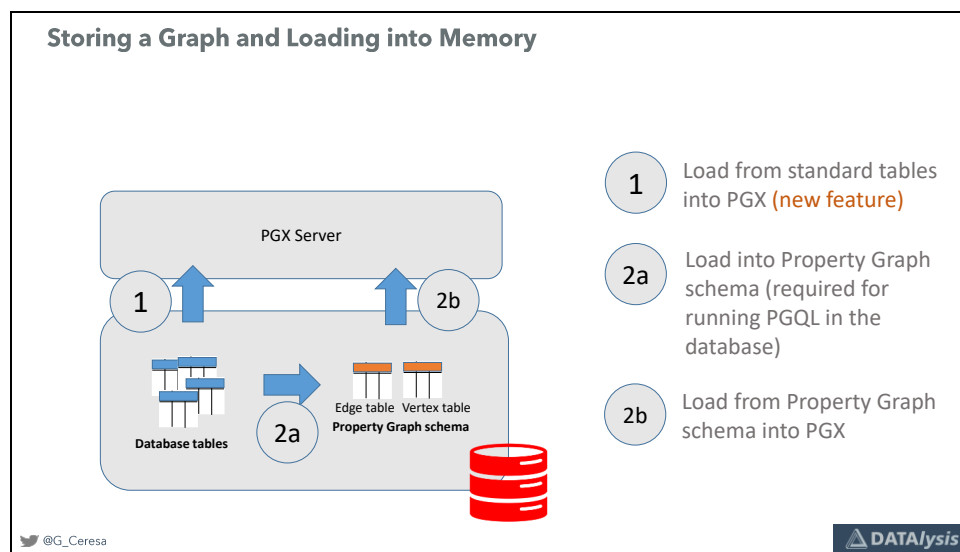


Oracle Property Graph can be used in 2 ways: the two tier approach which run graph queries directly on the Oracle database used for storing the graph. The graph query language, PGQL, is translated into standard SQL to provide the answer to the request.

The more classical approach, and the one giving access to the full power of Oracle property graph is by using PGX. A graph is loaded from the storage layer into PGX. Clients connect to PGX and perform PGQL queries or execute algorithms for their analysis. PGX can be executed as standalone process or by deploying it in an application server.



Slide 18



When loading a graph into PGX from a relational database, two different ways are not possible with the release of Graph server 20.1.0.

The "direct" (1) way is providing to PGX a configuration defining all the objects to load from database objects directly, defining what nodes and edges are, the various properties etc.

The "classical" (2) way of loading a graph is a 2-step activity: first a graph schema is created in the database. This schema is actually 5 tables with a fixed structure and indexes and partitioning already defined. You will load into these tables the content of the graph (2a). After you can tell PGX to load the graph (2b) by simply providing the graph name, database connection parameters and the list of properties you want to load for both nodes and edges (so you load only what you really need for a faster load time and a minimal memory usage).

This lab will follow the "classical" way, performing both 2a and 2b.



Slide 19

**Test connectivity & environments for Labs**

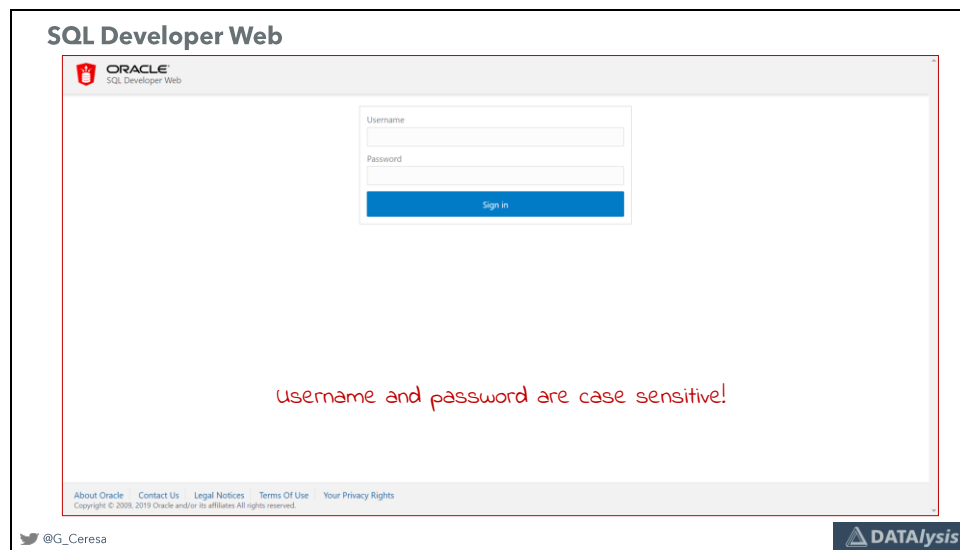
 @G\_Ceresa

 DATAlysis





Slide 20

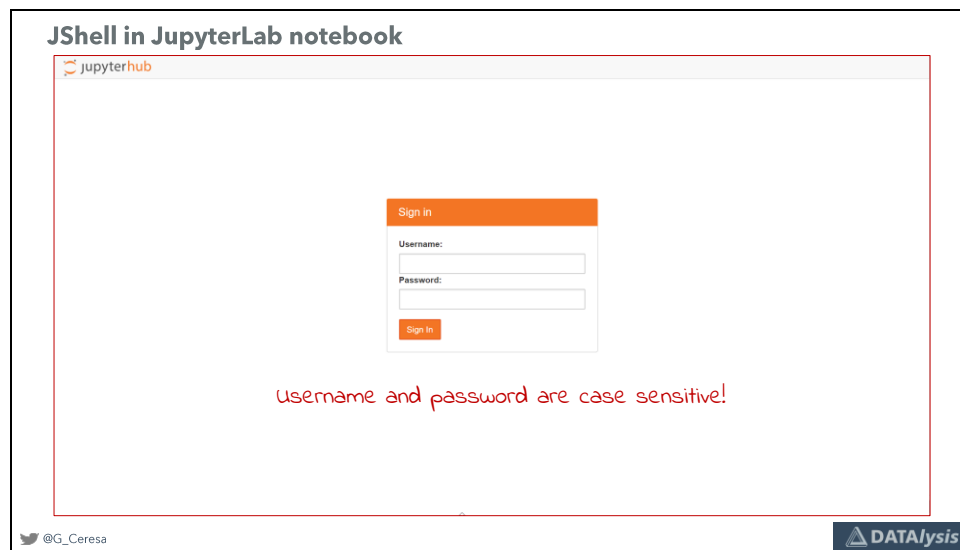


The lab uses an Autonomous Data warehouse in the cloud to store the graph. To connect SQL Developer web is used. It is available by default in the Autonomous Data warehouse cloud instance.

You can also use SQL Developer or any other tool able to perform SQL operations in the database.



## Slide 21



For the part related to PGX itself a JupyterLab notebook is used to provide a more friendly interface on top of the JShell PGX client. If you download the Graph client you have the JShell command line interface available directly (requires Java 11+). The notebook doesn't provide extra features for PGX itself, but has the advantage to allow to add comments in markdown etc.



Slide 22

Introducing "SH" and the scenario for the Labs

 @G\_Ceresa

 DATAlysis



## Slide 23

**Scenario used for the Labs**

Imagine you work for a web shop...

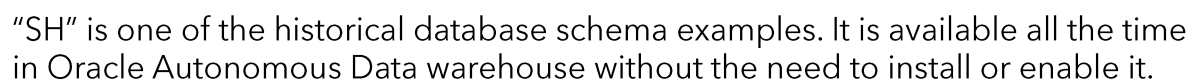
- You have products
- You have customers
- You have orders of products by customers
- Your customers are worldwide, you have their country
- You have sales channels
- You have promotions on products

From all that, you are going to create a graph and do queries on it.

 @G\_Ceresa

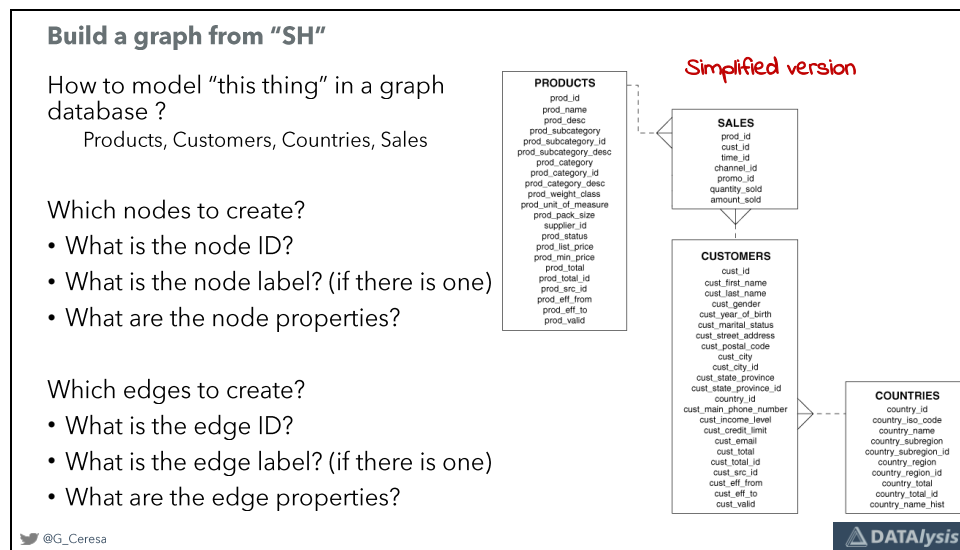
 DATAlysis

The lab scenario is a simple e-shop commerce activity.





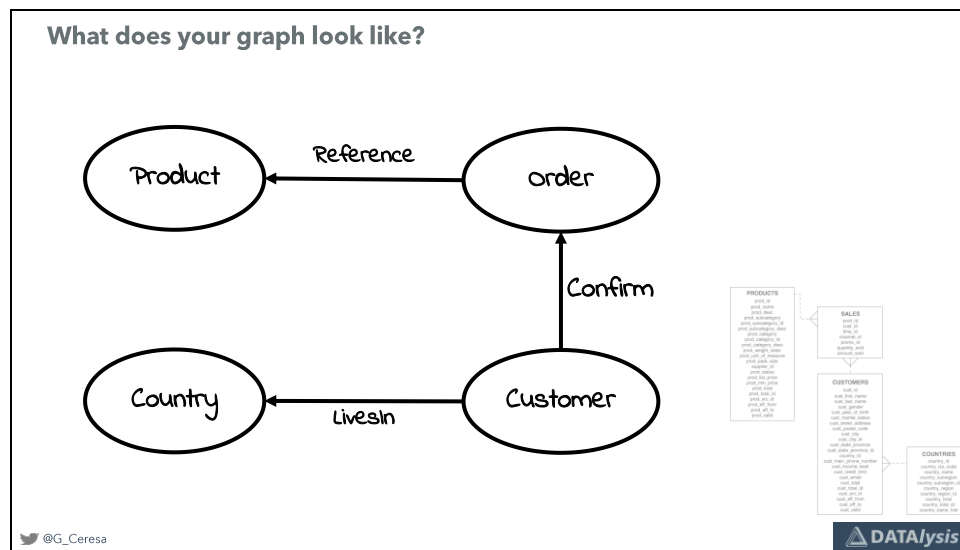
## Slide 25



To build a graph the challenge, like any other database or solution, is to find the right model.



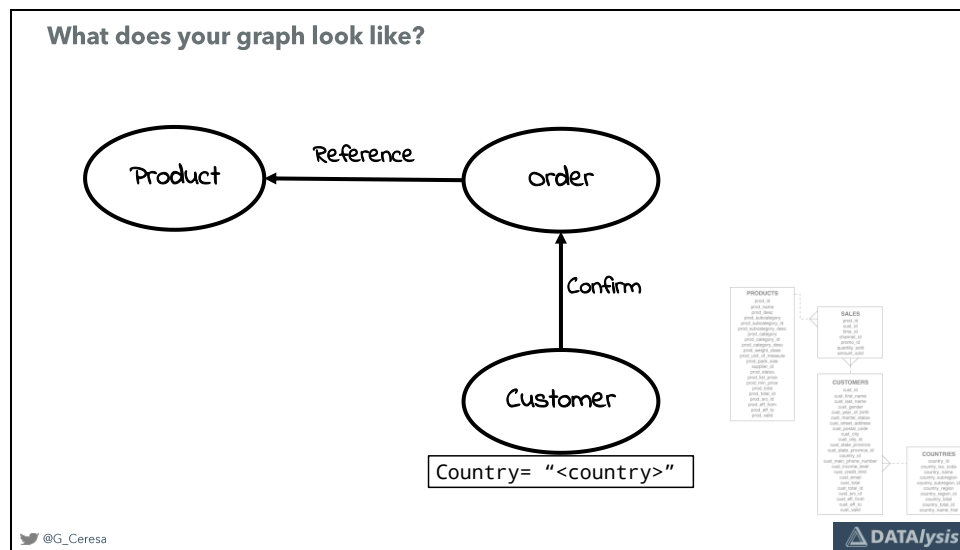
Slide 26



4 tables could be turned into a graph with a node for each row of each table.



Slide 27

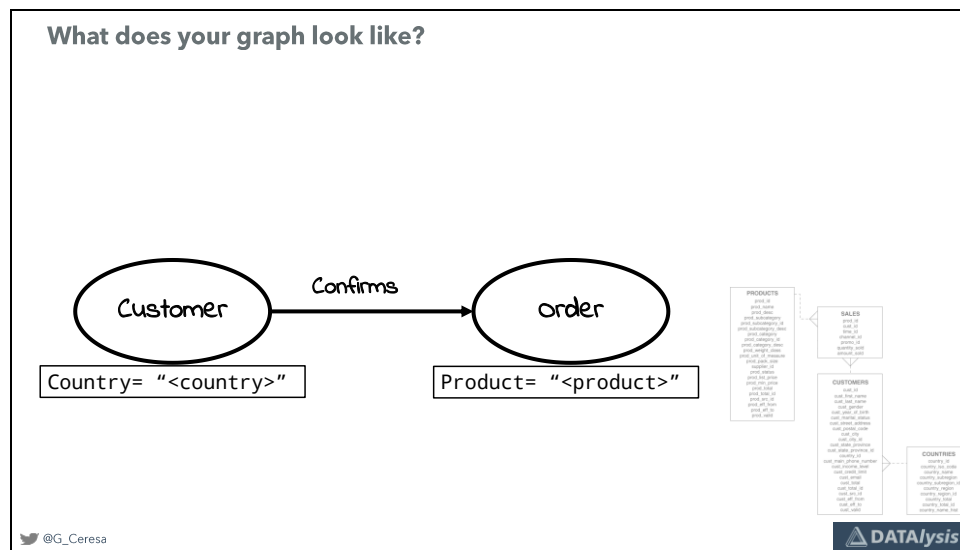


But the content of one of the tables could also be loaded as a property of another node.





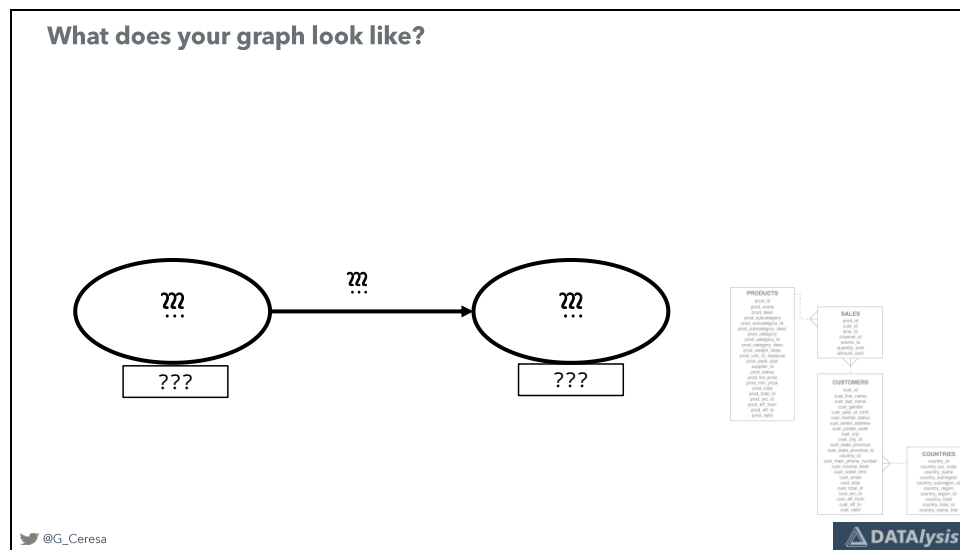
Slide 28



And the logic can be applied to reduce the graph even more.



Slide 29



All these models aren't wrong. Technically they are graph still. The question is: are they going to allow you to perform the analysis you need?

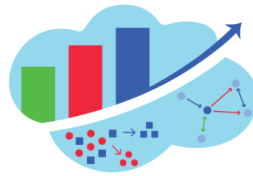


Slide 30

**Lab 1: design your graph**

 @G\_Ceresa

 DATAlysis



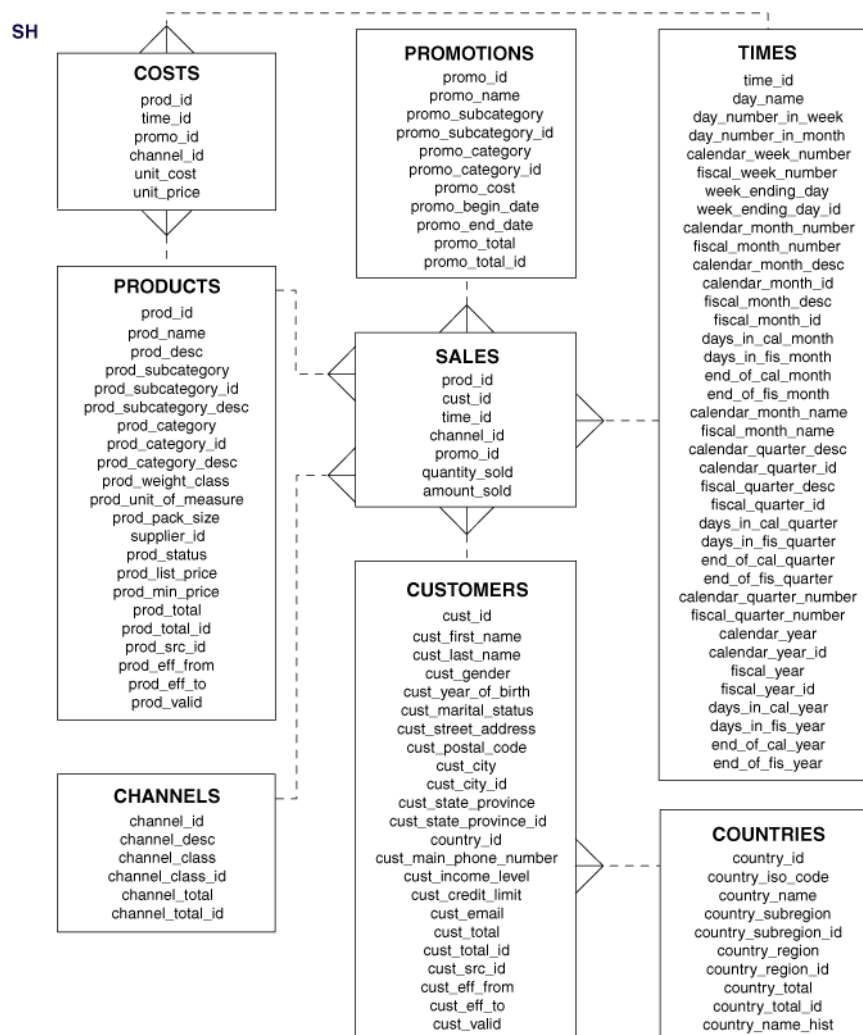
## Lab 1: Design your graph

### Target

In this Lab you will design possible graph structures derived from the scenario and the available sources.

The context of this lab is a simple sales activity, with customers buying products. It's one of the historical sample schemas you can have with any Oracle Database and it's also available by default in any Autonomous Database.

### Oracle Sales History (SH) schema diagram





Focusing on a reduced model covering only the tables "PRODUCTS", "SALES", "CUSTOMER" and "COUNTRIES", design on a piece of paper the possible structure of your graph in a generic way.

With a single node identified by its label and listing the properties keys. Add the edges connecting these nodes and identify the label and properties for the edges as well.

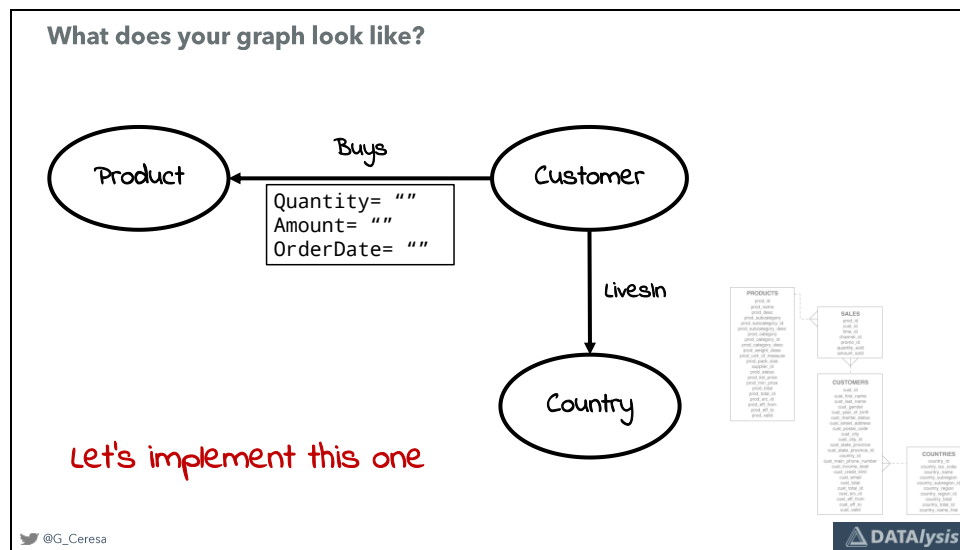
There isn't a right or wrong model as long as you respect the basic rules and technical constraints of a property graph.

But a model will be easier to query than another.

The expected usage drives the choice of the model, like when you design a data warehouse or any other relational database.



Slide 31



A model with seem well balanced for most of the needs is to load the “order lines” as edges, connecting the customer and various products composing each order.



Slide 32

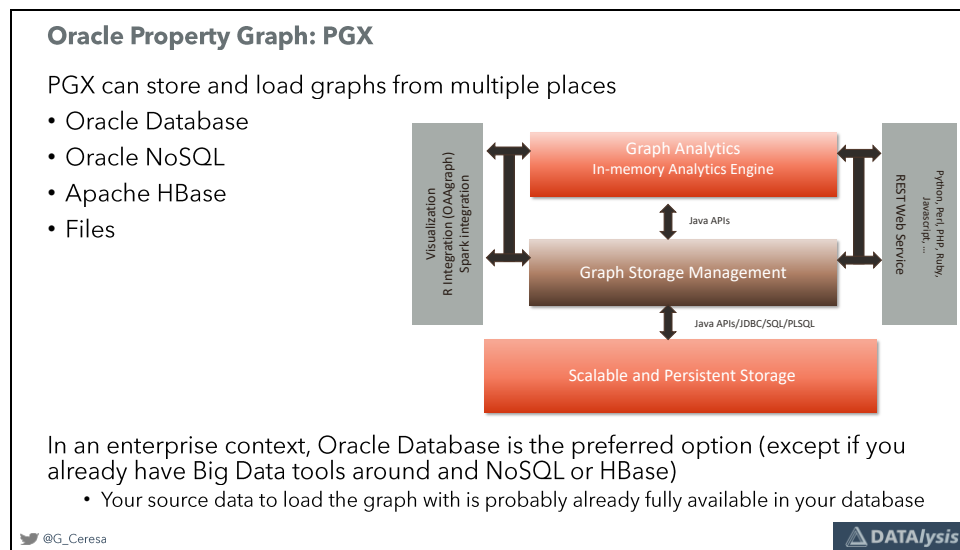
**Storing a Property Graph: options, Oracle Database and example**

 @G\_Ceresa

 DATAlysis



Slide 33



In an Enterprise environment we can exclude the storage in files, this one can work for sandboxing and testing, but not for a real usage.

Companies generally always have most of their data into a database, therefore it makes sense to store the graph there. This also allow to be compliant with security or auditing rules as these processed would already be known.





Slide 34

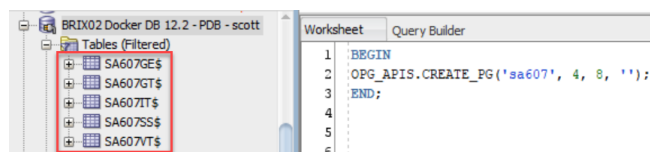
**Oracle Database - Create a new graph**

What you need:

- Oracle Database 12c R2 or newer (including Autonomous)
- Extended Data Types (to have varchar of more than 4'000)

**BEGIN**

```
OPG_APIS.CREATE_PG('name_of_your_graph');  
END;
```



GE\$ : edges of the graph  
VT\$ : vertices of the graph  
GT\$ : graph skeleton  
IT\$ : text index metadata  
SS\$ : graph snapshots

@G\_Ceresa

DATAlysis

To create a graph in the database you have to call a method. The mandatory parameter is the graph name, others are available. Reference to the documentation for all the details.

The result is 5 tables named like the graph name plus a suffix.



## Slide 35

## Oracle Database - Create a new graph

- The graph can be loaded by SQL, doing standard "INSERT" into the tables

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1 VID	NUMBER	No	(null)	1	Vertex ID
2 K	NVARCHAR2(3100 CHAR)	Yes	(null)	2	Property key
3 T	NUMBER(38,0)	Yes	(null)	3	Property value type
4 V	NVARCHAR2(15000 CHAR)	Yes	(null)	4	Property value (text)
5 VN	NUMBER	Yes	(null)	5	Property value - numeric
6 VT	TIMESTAMP(4) WITH TIME ZONE	Yes	(null)	6	Property value - date-time
7 SL	NUMBER	Yes	(null)	7	Security label
8 VTS	DATE	Yes	(null)	8	Validity start
9 VTE	DATE	Yes	(null)	9	Validity end
10 FE	NVARCHAR2(4000 CHAR)	Yes	(null)	10	Future extension

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1 EID	NUMBER	No	(null)	1	Edge ID
2 SVID	NUMBER	No	(null)	2	Source Vertex ID
3 DVID	NUMBER	No	(null)	3	Destination Vertex ID
4 EL	NVARCHAR2(3100 CHAR)	Yes	(null)	4	Edge label
5 K	NVARCHAR2(3100 CHAR)	Yes	(null)	5	Property key
6 T	NUMBER(38,0)	Yes	(null)	6	Property value type
7 V	NVARCHAR2(15000 CHAR)	Yes	(null)	7	Property value (text)
8 VN	NUMBER	Yes	(null)	8	Property value - numeric
9 VT	TIMESTAMP(4) WITH TIME ZONE	Yes	(null)	9	Property value - date-time
10 SL	NUMBER	Yes	(null)	10	Security label
11 VTS	DATE	Yes	(null)	11	Validity start
12 VTE	DATE	Yes	(null)	12	Validity end
13 FE	NVARCHAR2(4000 CHAR)	Yes	(null)	13	Future extension

The structure of the tables for vertices and edges is fairly similar. Content is stored by row, this means that every property will be a row, and the VID would be repeated for the nodes. For the edges it's the same thing but the 4 columns EID, SVID, DVID and EL would be repeated to store the properties on edges by row.



Slide 36

**Oracle Database - Create a new graph**

The graph is stored "by rows", like doing an UNPIVOT on the tables.

@G\_Ceresa **DATAlysis**

The sources used to populate the graph are generally storing information by columns (the various columns of a table). To load this into a graph an "UNPIVOT" operation must be performed turning columns into rows.



Slide 37

**Oracle Database - Create a new graph**

**Vertex Table: "<graph>VT\$"**

Name	Null?	Type
VID	NOT NULL	NUMBER
VL		NVARCHAR2(3100)
K		NVARCHAR2(3100)
T		INTEGER
V		NVARCHAR2(15000)
VN		NUMBER
VT		TIMESTAMP WITH TIMEZONE

47

name: Matthew McConaughey [T=1]  
age: 47 [T=2]  
birth-date: 1969-11-04 12:00:00.0 [T=5]

**Edge Table: "<graph>GE\$"**

Name	Null?	Type
EID	NOT NULL	NUMBER
SVID	NOT NULL	NUMBER
DVID	NOT NULL	NUMBER
EL		NVARCHAR2(3100)
K		NVARCHAR2(3100)
T		INTEGER
V		NVARCHAR2(15000)
VN		NUMBER
VT		TIMESTAMP WITH TIME ZONE

46 → 47

admires  
weight: 1.0 [T=4]

@G\_Ceresa **DATA**lysis

A visual representation, with matching colours, of what information is stored in which column of the tables.



Slide 38

**Oracle Database - Create a new graph**

Data Types:

- All numeric properties go in VN
- Date/time properties go in VT
- All others go in V
- Booleans are encoded as "Y" / "N"

NOTE: All numeric and date properties are also stored in V in printable format (to enable text indexing)

ID	Data type	Column
1	String	V
2	Integer	VN
3	Float	VN
4	Double	VN
5	Date	VT
6	Boolean	V
7	Long	VN
8	Short	VN
9	Byte	VN
10	Char	V
101	Serializable	V

@G\_Ceresa

The properties can have various types. To keep track of this information the column T is used, and the columns VN and VT with the appropriate data type exists.



Slide 39

**Oracle Database - Create a new graph**

- These tables are just “normal” tables, queries can be done ...
- Support some graph algorithms, the doc list all the supported methods  
[https://docs.oracle.com/en/database/oracle/oracle-database/18/spgdg/OPG\\_API-reference.html](https://docs.oracle.com/en/database/oracle/oracle-database/18/spgdg/OPG_API-reference.html)

The screenshot shows an Oracle SQL Worksheet with a 'Query Builder' tab. The query is as follows:

```
1  
2 select 'Graph has ' || count(distinct eid) || ' edges' from sa607ge$  
3 union all  
4 select 'Graph has ' || count(distinct vid) || ' vertices' from sa607vt$;  
5  
6
```

Below the query, the 'Query Result' tab shows the results of the query. The results are displayed in a table with the following columns: 'GRAPH HAS' and 'COUNT(DISTINCT ID) | 'EDGES' | 'VERTICES'.

	GRAPH HAS	COUNT(DISTINCT ID)   'EDGES'   'VERTICES'
1	Graph has 105500 edges	
2	Graph has 48155 vertices	

@G\_Ceresa

DATAlysis

Once a graph is loaded in the database (which is the storage layer in this case), normal queries can be performed as it's “just” tables in a database.



Slide 40

### Oracle Database - Create a new graph

**Worksheet Query Builder**

```
1
2 select distinct k, t from ss607vt6 order by 1;
3
4
```

**Query Result** All Rows Fetched: 43 in 0.467 seconds

	K	T
1	ObjAuditParseDate	5
2	caption	1
3	cn	1
4	creator	1
5	dataType	1
6	dbFlag	1
7	dbName	1
8	dbTypeId	1
9	description	1

**Worksheet Query Builder**

```
1
2 select t from ss607vt6 order by 1, 2;
3
4
```

**Query Result** Fetched 50 rows in 1.039 seconds

	VID	K	T	V	VS	VT	SL	VTS	VTE	FE
1	1	ObjAuditParseDate	5	Thu May 10 17:00:55 CEST 2018	(null)	10-MAY-2018 17:00:55	EUROPE/STWICH	(null)	(null)	(null)
2	1	fqn	1	QueryPriv_4204:878181286193911	(null)	(null)	(null)	(null)	(null)	(null)
3	1	keyDomain	1	RPD Security	(null)	(null)	(null)	(null)	(null)	(null)
4	1	keyId	1	QueryPriv_4204:878181286193911	(null)	(null)	(null)	(null)	(null)	(null)
5	1	maxExecTime	1	400	(null)	(null)	(null)	(null)	(null)	(null)
6	1	maxRows	1	100000	(null)	(null)	(null)	(null)	(null)	(null)
7	1	name	1	QueryPriv_4204:878181286193911	(null)	(null)	(null)	(null)	(null)	(null)
8	1	objectType	1	14204	(null)	(null)	(null)	(null)	(null)	(null)
9	1	xx id	1	14204:27160	(null)	(null)	(null)	(null)	(null)	(null)

@G\_Ceresa


**DATAlysis**


Clearly queries can be fairly simple or get quite complex if you try to perform real graph queries writing SQL yourself by hand.




Slide 41

Oracle Database - Create a new graph

Example 

 @G\_Ceresa







Slide 42

**Lab 2: create and populate a graph** @G\_Ceresa DATAlysis



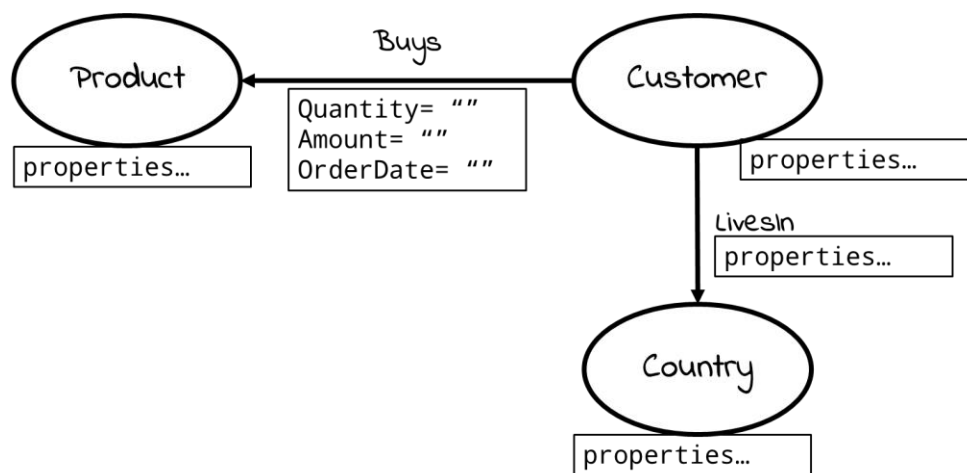
## Lab 2: Create and populate a graph

### Target

After this lab you will have in the Oracle Autonomous Datawarehouse relational database a property graph stored in the graph schema and with all the nodes, edges and properties populated with data.

For this Lab you will work only in SQL Developer Web, using the URL and credentials provided.

The graph you are going to create has the following structure.



1) To create the tables with the fixed structures to store the graph execute the following code.

```
BEGIN
  OPG_APIS.CREATE_PG('mysales');
END;
```

Refreshing the list of tables in your own schema you will see the newly created tables for the graph.



Oracle SQL Developer interface showing the execution of a PL/SQL procedure. The Navigator on the left shows the 'LABUSER00' schema with tables like MYSALES\$. The main window shows the execution of 'OPG\_API.CREATE\_PG('mysales');'. The status bar at the bottom indicates 'PL/SQL procedure successfully completed.'

2) Inspect the source data you are going to use to populate your graph by switching to the SH schema.

Oracle SQL Developer interface showing a query result for the SH schema. The Navigator on the left shows the 'SH' schema with tables like CHANNELS, COSTS, COUNTRIES, etc. The main window shows a query result table with columns: country\_id, country\_iso\_code, country\_name, country\_subregion, country\_subregion\_id, country\_region, country\_region\_id, country\_total\_id, country\_name\_hist. The status bar at the bottom indicates '23 rows total'.

	country_id	country_iso_code	country_name	country_subregion	country_subregion_id	country_region	country_region_id	country_total_id	country_name_hist
1	52771	CN	China	Asia	52793	Asia			
2	52781	IN	India	Asia	52793	Asia			
3	52782	JP	Japan	Asia	52793	Asia			
4	52783	MY	Malaysia	Asia	52793	Asia			
5	52769	SG	Singapore	Asia	52793	Asia			
6	52791	ZA	South Africa	Africa	52792	Africa			
7	52774	AU	Australia	Oceania	52794	Oceania			



3) Verify if the primary keys of the existing tables can be used as IDs for the nodes. Node's IDs are unique across the whole graph, while a table primary key is generally unique only in the table itself.

```
SELECT 'customer ID' as id, MIN(cust_id) as min_id, MAX(cust_id) as max_id, COUNT(DISTINCT cust_id) as unique_ids, COUNT(*) as nrows
FROM sh.customers
UNION ALL
SELECT 'product ID', MIN(prod_id), MAX(prod_id), COUNT(DISTINCT prod_id), COUNT(*) FROM sh.products
UNION ALL
SELECT 'country ID', MIN(country_id), MAX(country_id), COUNT(DISTINCT country_id), COUNT(*) FROM sh.countries;
```

The IDs of the 3 tables overlaps, they can't be used as IDs for nodes directly.

	id	min_id	max_id	unique_ids	nrows
1	customer ID	1	104500	55500	55500
2	product ID	13	148	72	72
3	country ID	52769	52791	23	23

4) One possible workaround for this problem is making the IDs unique adding an offset value by table.

```
SELECT 'customer ID' as id, MIN(cust_id) as min_id, MAX(cust_id) as max_id, COUNT(DISTINCT cust_id) as unique_ids, COUNT(*) as nrows
FROM sh.customers
UNION ALL
SELECT 'product ID', MIN(prod_id + 200000), MAX(prod_id + 200000), COUNT(DISTINCT prod_id), COUNT(*) FROM sh.products
UNION ALL
SELECT 'country ID', MIN(country_id + 300000), MAX(country_id + 300000), COUNT(DISTINCT country_id), COUNT(*) FROM sh.countries;
```



The IDs aren't overlapping anymore. This workaround is valid only during tests and must not be used in a real production environment as nothing prevent the IDs to keep growing and overlap again at some point.

Oracle SQL Developer interface showing a SQL query and its results.

Query:

```

1 SELECT 'customer ID' as id, MIN(cust_id) as min_id, MAX(cust_id) as max_id, COUNT(DISTINCT cust_id) as unique_ids, COUNT(*) as nrows FROM
2 sh.customers
3 UNION ALL
4 SELECT 'product ID', MIN(prod_id + 200000), MAX(prod_id + 200000), COUNT(DISTINCT prod_id), COUNT(*) FROM sh.products
5 UNION ALL
6 SELECT 'country ID', MIN(country_id + 300000), MAX(country_id + 300000), COUNT(DISTINCT country_id), COUNT(*) FROM sh.countries;

```

Query Result:

	id	min_id	max_id	unique_ids	nrows
1	customer ID	1	104500	55500	55500
2	product ID	200013	200148	72	72
3	country ID	352769	352791	23	23

5) Prepare the nodes representing the countries, performing the UNPIVOT to transform columns into rows and formatting the result to respect the graph rules. This is a validation step testing the query only.

```

SELECT country_id + 300000 as vid, 'label' as k, 1 as t, 'country'
as v, NULL as vn, NULL as vt FROM sh.countries
UNION ALL
SELECT country_id + 300000 as vid, 'name' as k, 1 as t, country_name
as v, NULL as vn, NULL as vt FROM sh.countries
UNION ALL
SELECT country_id + 300000 as vid, 'isoCode' as k, 1 as t,
country_iso_code as v, NULL as vn, NULL as vt FROM sh.countries
UNION ALL
SELECT country_id + 300000 as vid, 'sourceId' as k, 2 as t,
TO_CHAR(country_id) as v, country_id as vn, NULL as vt FROM
sh.countries
ORDER BY 1, 2;

```

Don't forget the offset added to the IDs, the "ORDER BY" is there to help highlighting the fact that a single node has multiple rows repeating the ID.



Oracle SQL Developer interface showing a SQL query and its results.

**SQL Query:**

```

1 SELECT country_id + 300000 as vid, 'label' as k, 1 as t, 'country' as v, NULL as vn, NULL as vt FROM sh.countries
2 UNION ALL
3 SELECT country_id + 300000 as vid, 'name' as k, 1 as t, country_name as v, NULL as vn, NULL as vt FROM sh.countries
4 UNION ALL
5 SELECT country_id + 300000 as vid, 'isoCode' as k, 1 as t, country_iso_code as v, NULL as vn, NULL as vt FROM sh.countries
6 UNION ALL
7 SELECT country_id + 300000 as vid, 'sourceId' as k, 2 as t, TO_CHAR(country_id) as v, country_id as vn, NULL as vt FROM sh.countries
8 ORDER BY 1, 2;

```

**Query Result:**

	vid	k	t	v	vn	vt
1	352769	isoCode	1	SG	(null)	(null)
2	352769	label	1	country	(null)	(null)
3	352769	name	1	Singapore	(null)	(null)
4	352769	sourceId	2	52769	52769	(null)
5	352770	isoCode	1	IT	(null)	(null)
6	352770	label	1	country	(null)	(null)
7	352770	name	1	Italy	(null)	(null)
8	352770	sourceId	2	52770	52770	(null)
9	352771	isoCode	1	CN	(null)	(null)

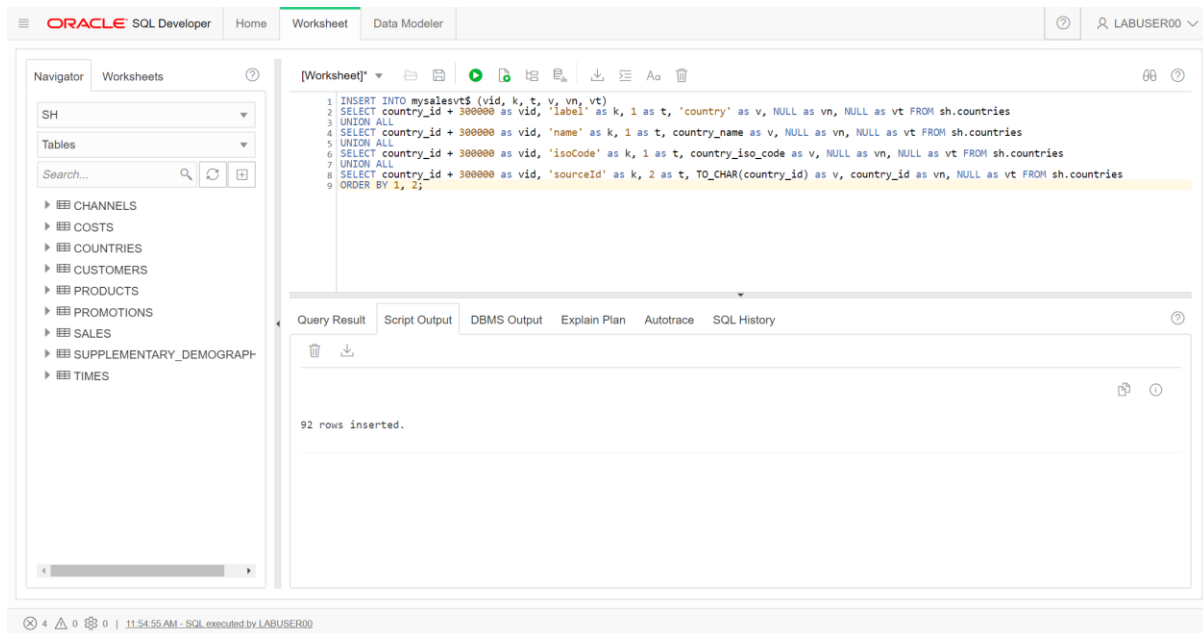
6.13.30 PM - 92 rows total

6) Insert the rows representing the countries nodes in the table.

```

INSERT INTO mysalestv$ (vid, k, t, v, vn, vt)
SELECT country_id + 300000 as vid, 'label' as k, 1 as t, 'country'
as v, NULL as vn, NULL as vt FROM sh.countries
UNION ALL
SELECT country_id + 300000 as vid, 'name' as k, 1 as t, country_name
as v, NULL as vn, NULL as vt FROM sh.countries
UNION ALL
SELECT country_id + 300000 as vid, 'isoCode' as k, 1 as t,
country_iso_code as v, NULL as vn, NULL as vt FROM sh.countries
UNION ALL
SELECT country_id + 300000 as vid, 'sourceId' as k, 2 as t,
TO_CHAR(country_id) as v, country_id as vn, NULL as vt FROM
sh.countries
ORDER BY 1, 2;

```



7) Prepare the nodes representing the products, performing the UNPIVOT to transform columns into rows and formatting the result to respect the graph rules. This is a validation step testing the query only.

```

SELECT prod_id + 200000 as vid, 'label' as k, 1 as t, 'product' as
v, NULL as vn, NULL as vt FROM sh.products
UNION ALL
SELECT prod_id + 200000 as vid, 'name' as k, 1 as t, prod_name as v,
NULL as vn, NULL as vt FROM sh.products
UNION ALL
SELECT prod_id + 200000 as vid, 'category' as k, 1 as t,
prod_category as v, NULL as vn, NULL as vt FROM sh.products
UNION ALL
SELECT prod_id + 200000 as vid, 'subcategory' as k, 1 as t,
prod_subcategory as v, NULL as vn, NULL as vt FROM sh.products
UNION ALL
SELECT prod_id + 200000 as vid, 'listPrice' as k, 3 as t,
TO_CHAR(prod_list_price) as v, prod_list_price as vn, NULL as vt
FROM sh.products
UNION ALL
SELECT prod_id + 200000 as vid, 'sourceId' as k, 2 as t,
TO_CHAR(prod_id) as v, prod_id as vn, NULL as vt FROM sh.products
ORDER BY 1, 2;
  
```

Don't forget the offset added to the IDs, the "ORDER BY" is there to help highlighting the fact that a single node has multiple rows repeating the ID.



Oracle SQL Developer interface showing a SQL query and its results.

**Query:**

```

1 SELECT prod_id + 200000 as vid, 'label' as k, 1 as t, 'product' as v, NULL as vn, NULL as vt FROM sh.products
2 UNION ALL
3 SELECT prod_id + 200000 as vid, 'name' as k, 1 as t, prod_name as v, NULL as vn, NULL as vt FROM sh.products
4 UNION ALL
5 SELECT prod_id + 200000 as vid, 'category' as k, 1 as t, prod_category as v, NULL as vn, NULL as vt FROM sh.products
6 UNION ALL
7 SELECT prod_id + 200000 as vid, 'subcategory' as k, 1 as t, prod_subcategory as v, NULL as vn, NULL as vt FROM sh.products
8 UNION ALL
9 SELECT prod_id + 200000 as vid, 'listPrice' as k, 3 as t, TO_CHAR(prod_list_price) as v, prod_list_price as vn, NULL as vt FROM sh.products
10 UNION ALL
11 SELECT prod_id + 200000 as vid, 'sourceId' as k, 2 as t, TO_CHAR(prod_id) as v, prod_id as vn, NULL as vt FROM sh.products
12 ORDER BY 1,2

```

**Query Result:**

	vid	k	t	v	vn	vt
1	200013	category	1	Photo	(null)	(null)
2	200013	label	1	product	(null)	(null)
3	200013	listPrice	3	899.99	899.99	(null)
4	200013	name	1	5MP Telephoto Dig...	(null)	(null)
5	200013	sourceId	2	13	13	(null)
6	200013	subcategory	1	Cameras	(null)	(null)
7	200014	category	1	Peripherals and Ac...	(null)	(null)
8	200014	label	1	product	(null)	(null)

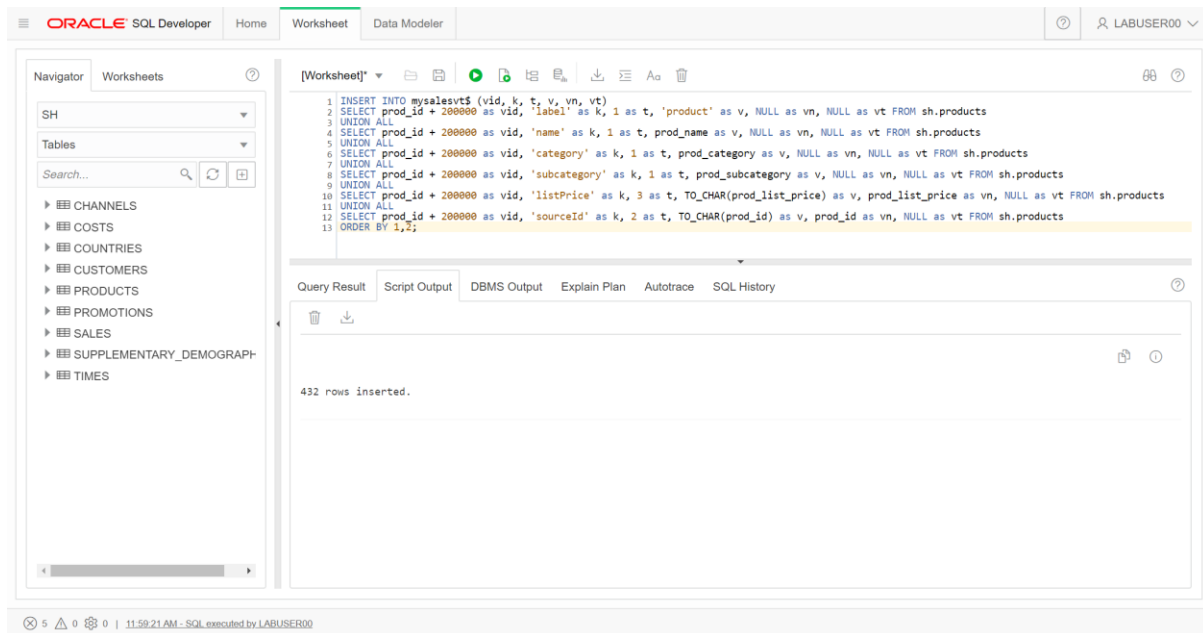
8) Insert the rows representing the products nodes in the table.

```

INSERT INTO mysalesvt$ (vid, k, t, v, vn, vt)
SELECT prod_id + 200000 as vid, 'label' as k, 1 as t, 'product' as
v, NULL as vn, NULL as vt FROM sh.products
UNION ALL
SELECT prod_id + 200000 as vid, 'name' as k, 1 as t, prod_name as v,
NULL as vn, NULL as vt FROM sh.products
UNION ALL
SELECT prod_id + 200000 as vid, 'category' as k, 1 as t,
prod_category as v, NULL as vn, NULL as vt FROM sh.products
UNION ALL
SELECT prod_id + 200000 as vid, 'subcategory' as k, 1 as t,
prod_subcategory as v, NULL as vn, NULL as vt FROM sh.products
UNION ALL
SELECT prod_id + 200000 as vid, 'listPrice' as k, 3 as t,
TO_CHAR(prod_list_price) as v, prod_list_price as vn, NULL as vt
FROM sh.products
UNION ALL
SELECT prod_id + 200000 as vid, 'sourceId' as k, 2 as t,
TO_CHAR(prod_id) as v, prod_id as vn, NULL as vt FROM sh.products
ORDER BY 1,2;

```





9) Prepare the nodes representing the customers, performing the UNPIVOT to transform columns into rows and formatting the result to respect the graph rules. This is a validation step testing the query only.

```
SELECT cust_id as vid, 'label' as k, 1 as t, 'customer' as v, NULL
as vn, NULL as vt FROM sh.customers
UNION ALL
SELECT cust_id as vid, 'name' as k, 1 as t, cust_first_name || ' '
|| cust_last_name as v, NULL as vn, NULL as vt FROM sh.customers
UNION ALL
SELECT cust_id as vid, 'gender' as k, 1 as t, cust_gender as v, NULL
as vn, NULL as vt FROM sh.customers
UNION ALL
SELECT cust_id as vid, 'maritalStatus' as k, 1 as t,
cust_marital_status as v, NULL as vn, NULL as vt FROM sh.customers
WHERE cust_marital_status IS NOT NULL
UNION ALL
SELECT cust_id as vid, 'yearOfBirth' as k, 2 as t,
TO_CHAR(cust_year_of_birth) as v, cust_year_of_birth as vn, NULL as
vt FROM sh.customers
UNION ALL
SELECT cust_id as vid, 'sourceId' as k, 2 as t, TO_CHAR(cust_id) as
v, cust_id as vn, NULL as vt FROM sh.customers
ORDER BY 1, 2;
```

The "ORDER BY" is there to help highlighting the fact that a single node has multiple rows repeating the ID. When a property isn't defined it is skipped, the graph doesn't deal well with NULL vs not defined.



The screenshot shows the Oracle SQL Developer interface. The 'Navigator' pane on the left shows a schema named 'SH' with various tables. The 'Worksheets' pane shows a SQL query. The 'Query Result' pane displays the results of the query, which is a table with 8 rows and 7 columns: vid, k, t, v, vn, vt.

vid	k	t	v	vn	vt
1	gender	1	M	(null)	(null)
2	label	1	customer	(null)	(null)
3	name	1	Abigail Kessel	(null)	(null)
4	sourceId	2	1	(null)	(null)
5	yearOfBirth	2	1946	(null)	(null)
6	gender	1	F	(null)	(null)
7	label	1	customer	(null)	(null)
8	name	1	Anne Koch	(null)	(null)

10) Insert the rows representing the customers nodes in the table.

```
INSERT INTO mysalestv$ (vid, k, t, v, vn, vt)
SELECT cust_id as vid, 'label' as k, 1 as t, 'customer' as v, NULL
as vn, NULL as vt FROM sh.customers
UNION ALL
SELECT cust_id as vid, 'name' as k, 1 as t, cust_first_name || ' '
|| cust_last_name as v, NULL as vn, NULL as vt FROM sh.customers
UNION ALL
SELECT cust_id as vid, 'gender' as k, 1 as t, cust_gender as v, NULL
as vn, NULL as vt FROM sh.customers
UNION ALL
SELECT cust_id as vid, 'maritalStatus' as k, 1 as t,
cust_marital_status as v, NULL as vn, NULL as vt FROM sh.customers
WHERE cust_marital_status IS NOT NULL
UNION ALL
SELECT cust_id as vid, 'yearOfBirth' as k, 2 as t,
TO_CHAR(cust_year_of_birth) as v, cust_year_of_birth as vn, NULL as
vt FROM sh.customers
UNION ALL
SELECT cust_id as vid, 'sourceId' as k, 2 as t, TO_CHAR(cust_id) as
v, cust_id as vn, NULL as vt FROM sh.customers
ORDER BY 1, 2;
```



```

1 INSERT INTO mysalesvt$ (vid, k, t, v, vn, vt)
2 SELECT cust_id as vid, 'label' as k, 1 as t, 'customer' as v, NULL as vn, NULL as vt FROM sh.customers
3 UNION ALL
4 SELECT cust_id as vid, 'name' as k, 1 as t, cust_first_name || ' ' || cust_last_name as v, NULL as vn, NULL as vt FROM sh.customers
5 UNION ALL
6 SELECT cust_id as vid, 'gender' as k, 1 as t, cust_gender as v, NULL as vn, NULL as vt FROM sh.customers
7 UNION ALL
8 SELECT cust_id as vid, 'maritalstatus' as k, 1 as t, cust_marital_status as v, NULL as vn, NULL as vt FROM sh.customers
9 WHERE cust_marital_status IS NOT NULL
10 UNION ALL
11 SELECT cust_id as vid, 'yearOfBirth' as k, 2 as t, TO_CHAR(cust_year_of_birth) as v, cust_year_of_birth as vn, NULL as vt FROM sh.customers
12 UNION ALL
13 SELECT cust_id as vid, 'sourceId' as k, 2 as t, TO_CHAR(cust_id) as v, cust_id as vn, NULL as vt FROM sh.customers
14 ORDER BY 1, 2;

```

Query Result: 315,572 rows inserted.

11) Execute some control queries on the nodes table to validate the content: number of nodes by label and number of nodes by property.

```

SELECT v, COUNT(DISTINCT vid) FROM mysalesvt$
WHERE k = 'label'
GROUP BY v
ORDER BY 1;

```

```

1 SELECT v, COUNT(DISTINCT vid) FROM mysalesvt$
2 WHERE k = 'label'
3 GROUP BY v
4 ORDER BY 1;

```

v	count(distinctvid)
1 country	23
2 customer	55500
3 product	72

```

SELECT k, COUNT(DISTINCT vid) FROM mysalesvt$
GROUP BY k
ORDER BY 2 DESC, 1;

```



The screenshot shows the Oracle SQL Developer interface. The left pane displays the Navigator with a tree view of database objects. The main pane shows a SQL query in the Worksheet:

```
1 SELECT k, COUNT(DISTINCT vid) FROM mysalesvt5
2 GROUP BY k
3 ORDER BY 2 DESC, 1;
```

The Query Result tab is active, displaying the following table:

	k	count(distinctvid)
1	label	55595
2	name	55595
3	sourceId	55595
4	gender	55500
5	yearOfBirth	55500
6	maritalStatus	38072
7	category	72
8	listPrice	72
9	subcategory	72
10	isoCode	23

12) For the edges there isn't an existing ID in the table, therefore you can't use that as base. The solution is to use a sequence in the database. It will keep track of the last value returned and keep increment this value to generate unique numbers. A sequence can be used for the nodes IDs as well to avoid the offset workaround.

```
CREATE SEQUENCE mysales_eid_seq;
```

The screenshot shows the Oracle SQL Developer interface. The left pane displays the Navigator. The main pane shows a SQL query in the Worksheet:

```
1 CREATE SEQUENCE mysales_eid_seq;
```

The Query Result tab is active, displaying the message:

Sequence MYSALES\_EID\_SEQ created.



13) Prepare the edges representing the  $(:customer) -[:livesIn] \rightarrow (:country)$  edges, performing the UNPIVOT to transform columns into rows and formatting the result to respect the graph rules. This is a validation step testing the query only.

```
SELECT NULL as eid
, cust_id as svid
, country_id + 300000 as dvid
, 'livesIn' as el
, 'stateProvince' as k
, 1 as t
, cust_state_province as v FROM sh.customers
ORDER BY 2;
```

Do not use the sequence when performing the validation step as it would waste numbers and time for nothing. Also don't forget the offset you have applied to the IDs of the countries.

The screenshot shows the Oracle SQL Developer interface. The SQL script in the worksheet is:

```
1 SELECT NULL as eid
2 , cust_id as svid
3 , country_id + 300000 as dvid
4 , 'livesIn' as el
5 , 'stateProvince' as k
6 , 1 as t
7 , cust_state_province as v FROM sh.customers
8 ORDER BY 2;
```

The Query Result tab displays the following data:

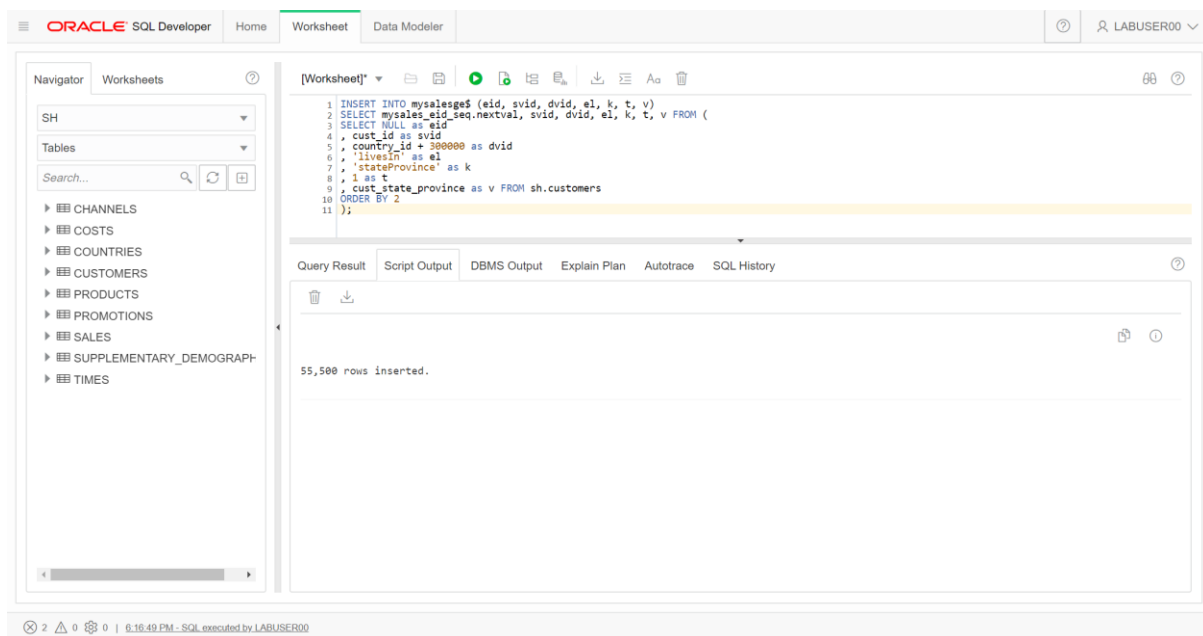
	eid	svid	dvid	el	k	t	v
1	(null)		1	352789	livesIn	stateProvince	1 England - N
2	(null)		2	352778	livesIn	stateProvince	1 Salamanca
3	(null)		3	352770	livesIn	stateProvince	1 Zeeland
4	(null)		4	352770	livesIn	stateProvince	1 Utrecht
5	(null)		5	352789	livesIn	stateProvince	1 England - N
6	(null)		6	352769	livesIn	stateProvince	1 Kuala Lumpi
7	(null)		7	352790	livesIn	stateProvince	1 HI
8	(null)		8	352790	livesIn	stateProvince	1 CO
9	(null)		9	352770	livesIn	stateProvince	1 Groningen

The status bar at the bottom indicates: 1 0 0 | 6:14:41 PM - 100 rows fetched, more to get



14) Insert the rows representing the (:customer) -[:livesIn]-> (:country) relationships into the table. Using the sequence this time, to generate a single IDs for every single edge.

```
INSERT INTO mysalesge$ (eid, svid, dvid, el, k, t, v)
SELECT mysales_eid_seq.nextval, svid, dvid, el, k, t, v FROM (
SELECT NULL as eid
, cust_id as svid
, country_id + 300000 as dvid
, 'livesIn' as el
, 'stateProvince' as k
, 1 as t
, cust_state_province as v FROM sh.customers
ORDER BY 2
);
```





15) The rows representing the edges (:customer) -[:buys]-> (:product) require an extra step. There are multiple attributes you want to collect as properties, these will be multiple rows in the table. But these rows must all share the same edge ID generated by the sequence. For this a temporary table will be created where an ID using the sequence is going to be assigned to every entry before to unpivot its columns. This is a validation query to check what the temporary table will look like.

```
SELECT NULL as eid
, cust_id as svid
, prod_id + 200000 as dvid
, 'buys' as el
, SUM(quantity_sold) as quantity_sold
, SUM(amount_sold) as amount_sold
, time_id as order_date FROM sh.sales
WHERE time_id >= to_date('20000101', 'yyyymmdd')
GROUP BY cust_id, prod_id, time_id;
```

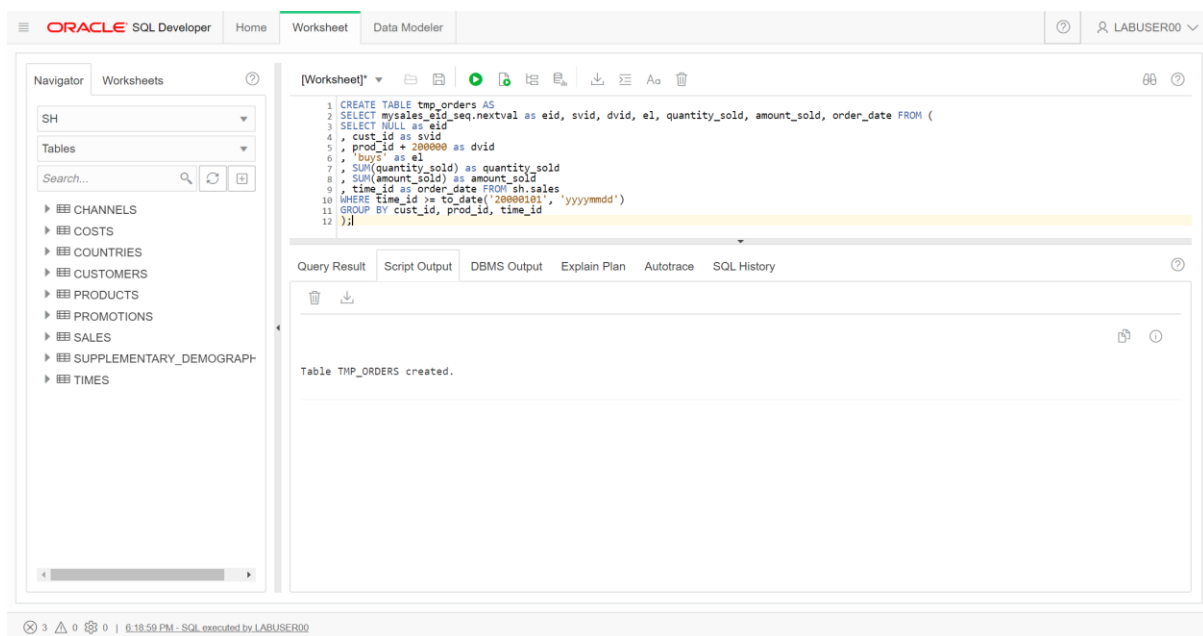
Do not forget to apply the offset on the products IDs. The query has a filter to only take half of the sales to keep the graph to a smaller size to make it faster for the lab environment.

	eid	svid	dvid	el	quantity_sold	amount_sold	order_date	
1	(null)		1848	200013	buys	1	1075.12	01/30/00 12:
2	(null)		7655	200013	buys	2	2150.24	01/30/00 12:
3	(null)		7759	200013	buys	2	2150.24	01/30/00 12:
4	(null)		3841	200013	buys	2	2136.08	02/03/00 12:
5	(null)		6751	200013	buys	1	1052.99	02/10/00 12:
6	(null)		7205	200013	buys	1	1052.99	02/10/00 12:
7	(null)		3045	200013	buys	2	2121.33	02/18/00 12:
8	(null)		2204	200013	buys	2	2136.08	02/21/00 12:
9	(null)		6116	200013	buys	1	1067.73	02/21/00 12:



```
CREATE TABLE tmp_orders AS
SELECT mysales_eid_seq.nextval as eid, svid, dvid, el,
quantity_sold, amount_sold, order_date FROM (
SELECT NULL as eid
, cust_id as svid
, prod_id + 200000 as dvid
, 'buys' as el
, SUM(quantity_sold) as quantity_sold
, SUM(amount_sold) as amount_sold
, time_id as order_date FROM sh.sales
WHERE time_id >= to_date('20000101', 'yyyymmdd')
GROUP BY cust_id, prod_id, time_id
);
```

The temporary table create used the sequence to populate the “eid” column with unique IDs.



```
SELECT * FROM tmp_orders;
```





ORACLE SQL Developer

Home Worksheet Data Modeler

LABUSER00

Navigator Worksheets

SH

Tables

Search...

CHANNELS

COSTS

COUNTRIES

CUSTOMERS

PRODUCTS

PROMOTIONS

SALES

SUPPLEMENTARY\_DEMOGRAPHICS

TIMES

[Worksheet] SELECT \* FROM tmp\_orders;

Query Result Script Output DBMS Output Explain Plan Autotrace SQL History

Download

	eid	svid	dvid	el	quantity_sold	amount_sold	order_date
1	55501	1848	200013	buys	1	1075.12	01/30/00 12:00:00
2	55502	7655	200013	buys	2	2150.24	01/30/00 12:00:00
3	55503	7759	200013	buys	2	2150.24	01/30/00 12:00:00
4	55504	3841	200013	buys	2	2136.08	02/03/00 12:00:00
5	55505	6751	200013	buys	1	1052.99	02/10/00 12:00:00
6	55506	7205	200013	buys	1	1052.99	02/10/00 12:00:00
7	55507	3045	200013	buys	2	2121.33	02/18/00 12:00:00
8	55508	2204	200013	buys	2	2136.08	02/21/00 12:00:00
9	55509	6116	200013	buys	1	1067.73	02/21/00 12:00:00

4 0 0 | 6:19:32 PM - 100 rows fetched, more to get

16) Prepare the edges representing the (:customer) -[:buys]-> (:product) relationships using the temporary table.

```
SELECT eid, svid, dvid, el, 'quantity' as k, 3 as t,
TO_CHAR(quantity_sold) as v, quantity_sold as vn, NULL as vt FROM
tmp_orders
UNION ALL
SELECT eid, svid, dvid, el, 'amount' as k, 3 as t,
TO_CHAR(amount_sold) as v, amount_sold as vn, NULL as vt FROM
tmp_orders
UNION ALL
SELECT eid, svid, dvid, el, 'orderDate' as k, 5 as t,
TO_CHAR(order_date, 'YYYY-MM-DD') as v, NULL as vn, order_date as vt
FROM tmp_orders
ORDER BY 1,2,3,4,5;
```



Oracle SQL Developer interface showing a SQL query and its results.

**Query:**

```

1 SELECT eid, svid, dvid, el, 'quantity' as k, 3 as t, TO_CHAR(quantity_sold) as v, quantity_sold as vn, NULL as vt FROM tmp_orders
2 UNION ALL
3 SELECT eid, svid, dvid, el, 'amount' as k, 3 as t, TO_CHAR(amount_sold) as v, amount_sold as vn, NULL as vt FROM tmp_orders
4 UNION ALL
5 SELECT eid, svid, dvid, el, 'orderDate' as k, 5 as t, TO_CHAR(order_date, 'YYYY-MM-DD') as v, NULL as vn, order_date as vt FROM tmp_orders
6 ORDER BY 1,2,3,4,5;

```

**Query Result:**

	eid	svid	dvid	el	k	t	v
1		55501	1848	200013	buys	amount	3 1075.12
2		55501	1848	200013	buys	orderDate	5 2000-01-30
3		55501	1848	200013	buys	quantity	3 1
4		55502	7655	200013	buys	amount	3 2150.24
5		55502	7655	200013	buys	orderDate	5 2000-01-30
6		55502	7655	200013	buys	quantity	3 2
7		55503	7759	200013	buys	amount	3 2150.24
8		55503	7759	200013	buys	orderDate	5 2000-01-30
9		55503	7759	200013	buys	quantity	3 2

17) Insert the rows representing the (:customer) -[:buys]-> (:product) relationships.

```

INSERT INTO mysalesge$ (eid, svid, dvid, el, k, t, v, vn, vt)
SELECT eid, svid, dvid, el, 'quantity' as k, 3 as t,
TO_CHAR(quantity_sold) as v, quantity_sold as vn, NULL as vt FROM
tmp_orders
UNION ALL
SELECT eid, svid, dvid, el, 'amount' as k, 3 as t,
TO_CHAR(amount_sold) as v, amount_sold as vn, NULL as vt FROM
tmp_orders
UNION ALL
SELECT eid, svid, dvid, el, 'orderDate' as k, 5 as t,
TO_CHAR(order_date, 'YYYY-MM-DD') as v, NULL as vn, order_date as vt
FROM tmp_orders
ORDER BY 1,2,3,4,5;

```



Oracle SQL Developer interface showing a SQL script in the Worksheet tab. The script is as follows:

```

1 INSERT INTO mysalesge$ (eid, svid, dvid, el, k, t, v, vn, vt)
2 SELECT eid, svid, dvid, el, 'quantity' as k, 3 as t, TO_CHAR(quantity_sold) as v, quantity_sold as vn, NULL as vt FROM tmp_orders
3 UNION ALL
4 SELECT eid, svid, dvid, el, 'amount' as k, 3 as t, TO_CHAR(amount_sold) as v, amount_sold as vn, NULL as vt FROM tmp_orders
5 UNION ALL
6 SELECT eid, svid, dvid, el, 'orderDate' as k, 5 as t, TO_CHAR(order_date, 'YYYY-MM-DD') as v, NULL as vn, order_date as vt FROM tmp_orders
7 ORDER BY 1,2,3,4,5;

```

The Query Result tab shows: 1,051,380 rows inserted.

18) Check the edges generated, counting them by label and property.

```

SELECT el, COUNT(DISTINCT eid) FROM mysalesge$
GROUP BY el
ORDER BY 1;

```

Oracle SQL Developer interface showing the same SQL script. The Query Result tab displays the following data:

el	count(distincteid)
1 buys	350460
2 livesIn	55500

```

SELECT k, COUNT(DISTINCT eid) FROM mysalesge$
GROUP BY k
ORDER BY 2 DESC, 1;

```



Oracle SQL Developer interface showing a query result table. The query is:

```
1 SELECT k, COUNT(DISTINCT eid) FROM mysaleses$
2 GROUP BY k
3 ORDER BY 2 DESC, 1;
```

The result table has the following data:

	k	count(distincteid)
1	amount	350460
2	orderDate	350460
3	quantity	350460
4	stateProvince	55500

19) Clean up the temporary table and also the sequence as the graph is fully loaded.

```
DROP TABLE tmp_orders;
```

```
DROP SEQUENCE mysales_eid_seq;
```

Oracle SQL Developer interface showing the execution of cleanup queries. The queries are:

```
1 DROP TABLE tmp_orders;
2
3 DROP SEQUENCE mysales_eid_seq;
```

The Query Result tab shows the following messages:

```
Table TMP_ORDERS dropped.

Sequence MYSALES_EID_SEQ dropped.
```



Slide 43

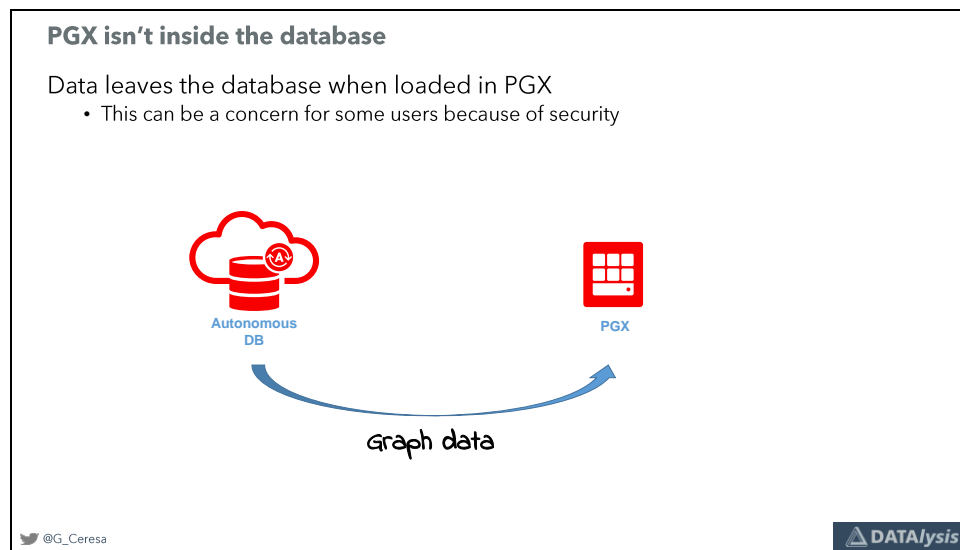
**PGX, loading graphs, PGQL, algorithms**

 @G\_Ceresa

 **DATAlysis**



Slide 44



PGX, the “engine” powering property graph in Oracle, is a separate process than the database. This means that when a graph is loaded in PGX, the data is leaving the database to be loaded into PGX:



Slide 45

**Loading a graph in PGX**

PGX is written in Java, to load a graph you have to use the existing Java packages

- The JavaDoc is the “bible”

Steps to load a graph (from any source):

- Create a *GraphConfigBuilder*, defining where to find the graph data
- Define all the properties for both nodes and edges you want to load  
(By default only the structure is loaded, no properties)
- “Build” the configuration  
(It will generate a JSON with all the settings)
- Call *readGraphWithProperties* passing the configuration JSON (can also be the path of a file containing the JSON)

It is not extremely intuitive, but still quite simple and there are examples provided with Oracle Graph Server 20.1.0 you can simply copy/paste the code from.

@G\_Ceresa

The process to load a graph in PGX is mainly creating a configuration defining where the graph is stored. Define which properties you want to load. Build the configuration which simply takes all the settings and some defaults to generate a Json string. And finally you can load the graph by using that Json (or a path pointing to a file containing the Json).



Slide 46

**Loading a graph in PGX**

Example of code:

```
var cfg = GraphConfigBuilder.forPropertyGraphRdbms()  
    .setName("graph_name")  
    .setJdbcUrl("jdbc:oracle:thin:@DB_host:DB_port/DB_service")  
    .setKeystoreAlias("") // this must be set or you get an error  
    .setUsername("DB_username")  
    .setPassword("DB_password")  
    .build();  
  
var graph = session.readGraphWithProperties(cfg);
```

@G\_Ceresa

A practical example looks like this.





Slide 47

**Loading a graph in PGX**

For any property you want to load, on both nodes or edges, you must explicitly declare it in the *GraphConfigBuilder*.

```
.addEdgeProperty("property_name", PropertyType.STRING)  
.addVertexProperty("property_name", PropertyType.DOUBLE)
```

The supported types (PropertyType) are:

BOOLEAN, DOUBLE, EDGE, FLOAT, INTEGER, LOCAL\_DATE, LONG,  
POINT2D, STRING, TIME, TIME\_WITH\_TIMEZONE, TIMESTAMP,  
TIMESTAMP\_WITH\_TIMEZONE, VERTEX

(the JavaDoc is the reference)

 @G\_Ceresa

 DATAanalysis

For the properties you will have to define them one by one, setting their name and type.



## Slide 48

**Loading a graph in PGX**

With a graph stored in the database you can generate the commands to load all the properties directly in the database

- Select from the \$GE and \$VT table all the properties keys and their type, build the full command and concatenate all with LISTAGG

```
WITH properties AS (  
  SELECT DISTINCT k, t, 'Vertex' AS kind FROM mysalesvt$  
  UNION ALL  
  SELECT DISTINCT k, t, 'Edge' AS kind FROM mysalesge$  
) , cfg AS (  
  SELECT '.add' || kind || 'Property("' || k || '",PropertyType.' ||  
  CASE  
    WHEN t = 1 THEN 'STRING' WHEN t = 2 THEN 'INTEGER' WHEN t = 3 THEN 'FLOAT'  
    WHEN t = 4 THEN 'DOUBLE' WHEN t = 7 THEN 'LONG' WHEN t = 5 THEN 'LOCAL_DATE'  
    WHEN t = 6 THEN 'BOOLEAN'  
  END || ')" AS prop  
  FROM properties  
  WHERE k IS NOT NULL  
) SELECT LISTAGG(prop, '') WITHIN GROUP(ORDER BY prop) FROM cfg;
```

@G\_Ceresa

DATAanalysis

When using a database as storage for the graph you can generate the whole list of configuration for all the properties by a SQL query on your database.



Slide 49

### Querying the graph

Once you have a graph in PGX, one of the activities you are probably going to do is querying the graph.

Oracle developed a language called PGQL : Property Graph Query Language

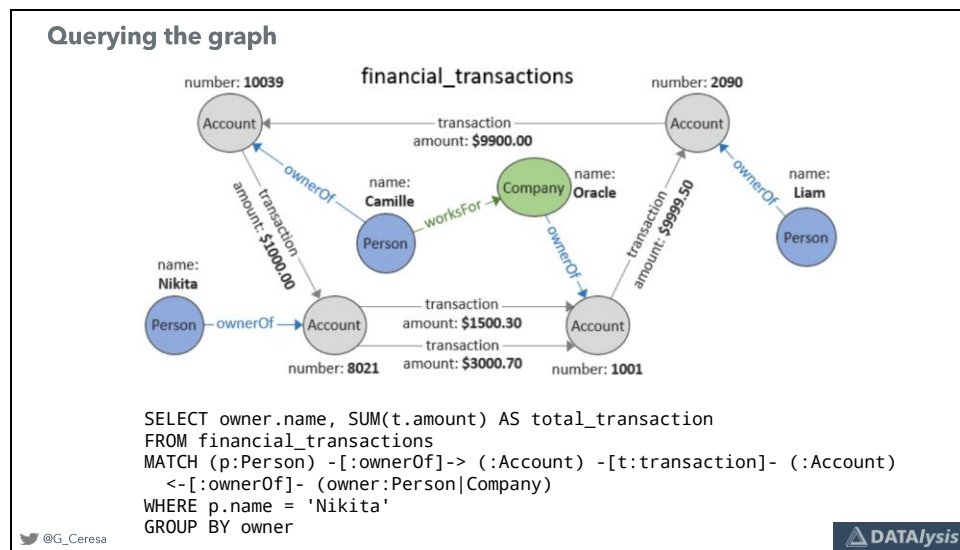
- It's supposed to be an extension of SQL adding graph specific syntax for pattern matching and other graph activities
- It's fairly young, support only a limited amount of functions and features of SQL
- It's visual being ASCII art-like
- It isn't a standard, it's open source but nobody else is using it
- It's going to be replaced (probably) at some point as a proper graph extension to the standard SQL is undergoing voting at ISO

@G\_Ceresa

Loading a graph is nice but useless if you don't use it. The most common thing is querying the graph by using PGQL, the property graph query language.



Slide 50



An example of PGQL on a sample graph: you can easily follow the logic of the MATCH condition in the above graph.

You can find all the details about PGQL at <http://pgql-lang.org/>.



Slide 51

**Querying the graph**

Practically in JShell it will be done in this way:

```
var pgql = "SELECT c.name, p.name, b.orderDate, b.amount,  
b.quantity WHERE (c:customer) -[b:buys]-> (p:product) LIMIT  
10";
```

```
PgqlResultSet resultSet = graph.queryPgql(pgql);
```

```
for (var result : resultSet) {  
    print(result.getString(1) + " bought " +  
          result.getString("p.name") + ": qty=" +  
          result.getFloat("b.quantity") + " on " +  
          result.getDate(3));  
}
```

@G\_Ceresa

In the PGX client JShell interface you will need to follow the rules of Java, the language used to interact with PGX in the JShell.



Slide 52

### Executing Graph algorithms

Querying the graph is one thing, but one of the real benefits with a graph are all the graph algorithms. They can provide answers to complex questions rapidly.

PGX comes with a set of about 60 algorithms available out of the box.

You can write new algorithms:

- Green-Marl
- Java

For many the source code is provided, making it easy to customize.

#### Built-In Algorithms

PGX includes a wide selection of optimized graph algorithms that can be invoked through the Analyst. The following table provides an overview of the available algorithms, grouped by category.

Category	Algorithms
Classic graph algorithms	Prim's Algorithm
Community detection	Conductance Minimization (Boman and Narang Algorithm), Infomap, Label Propagation
Connected components	Strongly Connected Components, Weakly Connected Components (WCC)
Link prediction	WTF (Whom To Follow) Algorithm
Matrix factorization	Matrix Factorization
Other	Graph Traversal Algorithms
Path finding	Bellman-Ford Algorithms, Bidirectional Dijkstra Algorithms, Dijkstra Algorithms, Fastest Path, Hop Distance Algorithms
Ranking and walking	Closeness Centrality Algorithms, Degree Centrality Algorithms, Eigenvector Centrality, Hyperlink-Induced Topic Search (HITS), PageRank Algorithms, Random Walk with Restart, Stochastic Approach for Link-Structure Analysis (SALSA) Algorithms, Vertex Betweenness Centrality Algorithms
Structure evaluation	Adamic-Adar index, Conductance, Cycle Detection Algorithms, Degree Distribution Algorithms, Eccentricity Algorithms, K-Core, Local Clustering Coefficient (LCC), Modularity, Partition Conductance, Reachability Algorithms, Topological Ordering Algorithms, Triangle Counting

@G\_Ceresa

DATAlysis

Another common activity is to execute algorithms. Oracle PGX comes with many algorithms available out of the box. You can also write your own if you need.



Slide 53

**Executing Graph algorithms**

You can't execute algorithms in a PGQL query.

You must first execute the algorithm on the graph and after query via PGQL for the results generated.

- Algorithms in general add new properties to the graphs elements.
- These properties contains the result of the algorithm.
- These changes aren't persisted, they are created in a copy of the original graph which is available only to the user who executed the algorithms.
- Once done with the analysis the changes are lost if nothing explicit is done to store them somewhere.

 @G\_Ceresa

Algorithms aren't executed via a PGQL query. You execute them in an "analyst" session and you then query the graph to get the result of the algorithms.



Slide 54

### Executing Graph algorithms

An example of PageRank calculation on a graph:

```
var analyst = session.createAnalyst();

var pagerank = analyst.pagerank(graph2);
print(pagerank);

var query = "SELECT v, v.name, v."+pagerank.getName()+" WHERE (v:product) ORDER BY v."+pagerank.getName()+" DESC LIMIT 10";
print(query);
var resultSet = graph2.queryPgql(query);
for (var result : resultSet) {
    print("node: "+result.getString(2)+" has pagerank = "+result.getDouble(3));
}
resultSet.close();
```

@G\_Ceresa

An example code to execute a page rank algorithm. Most algorithms add temporary properties to nodes and/or edges in the graph holding the results. The name of these properties can be retrieved by reading it from the returned value at the executing of the algorithm.





Slide 55

**Lab 3: load a graph, query it, run algorithms**

 @G\_Ceresa

 DATAlysis



## Lab 3: Load graph in PGX and query it

### Target

In this lab you will load a graph stored in a database into PGX, run PGQL queries and execute algorithms on it.

The Oracle Graph client comes with either a Groovy or JShell command line. For this Lab you will use a JupyterLab notebook on top of the Java Oracle Graph client.

The syntax and commands you can use are Java, they will work in the same way (or very close to) in the client itself.

The credentials for JupyterLab are the one provided.

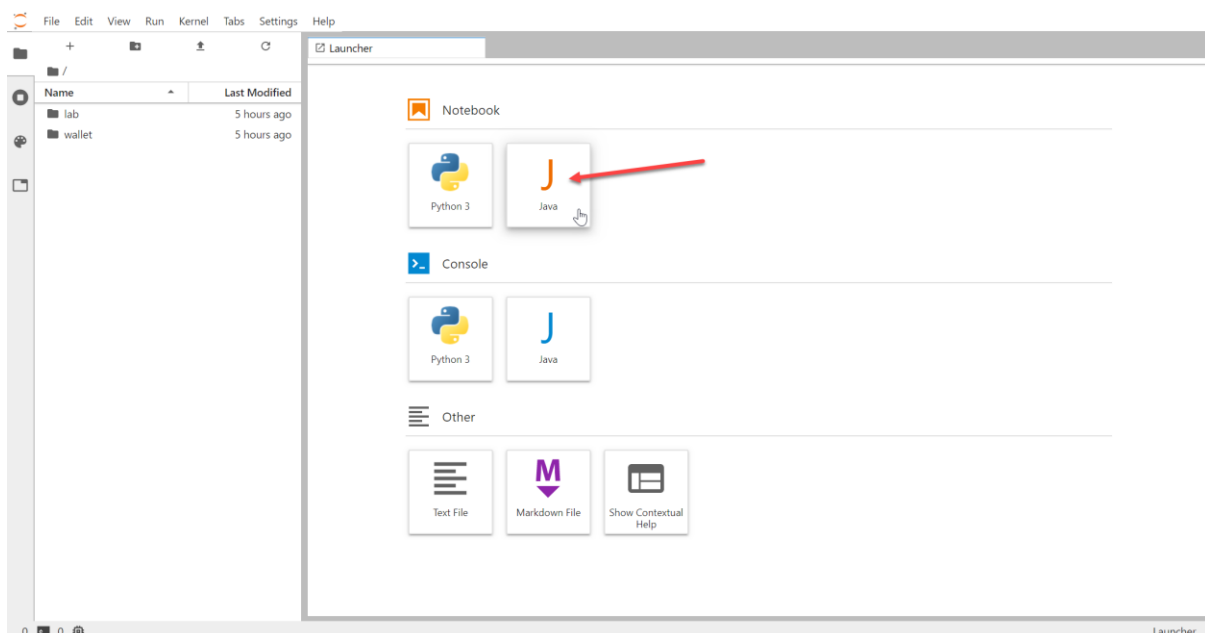
### Basic syntax rules

Every variable must be defined either with the real type (int, String, PgxSession etc.) or with the generic "var" keyword.

Commands must end with a ';', the last command or single-command cells will also work, but it's better to follow the rule to always end a command with ';'.

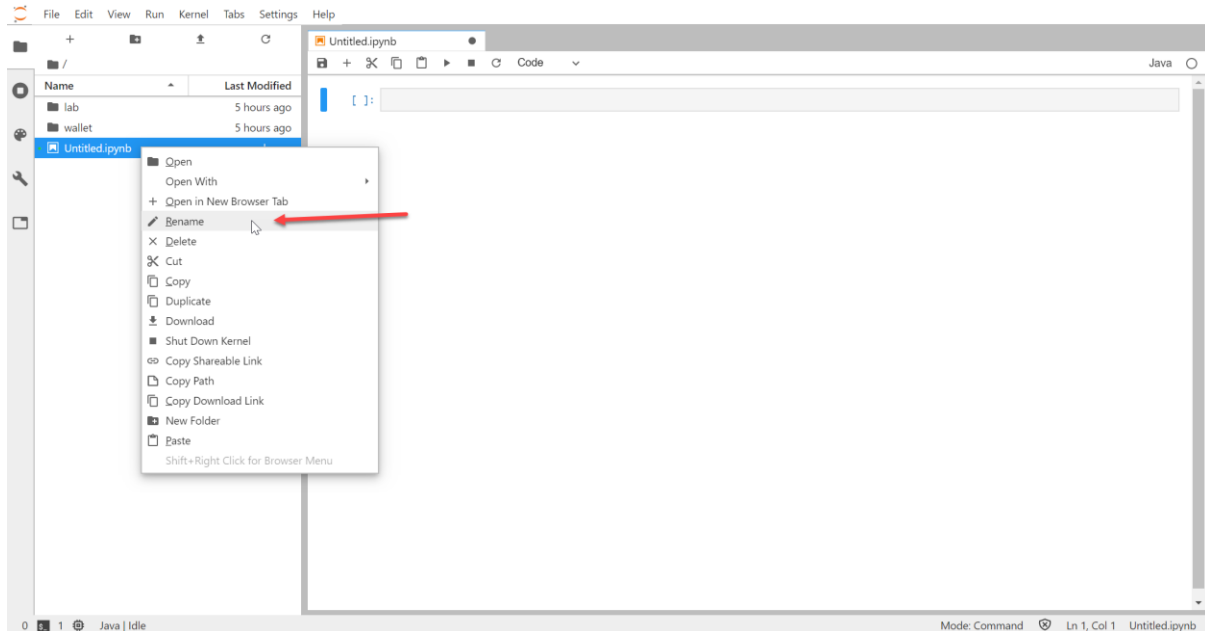
1) Create a new notebook based on Java.

Python could be used as well, but it would add a second layer of complexity with Python on top of Java using JPyte. In this Lab you are going to stick to Java which is as close the PGX as possible.



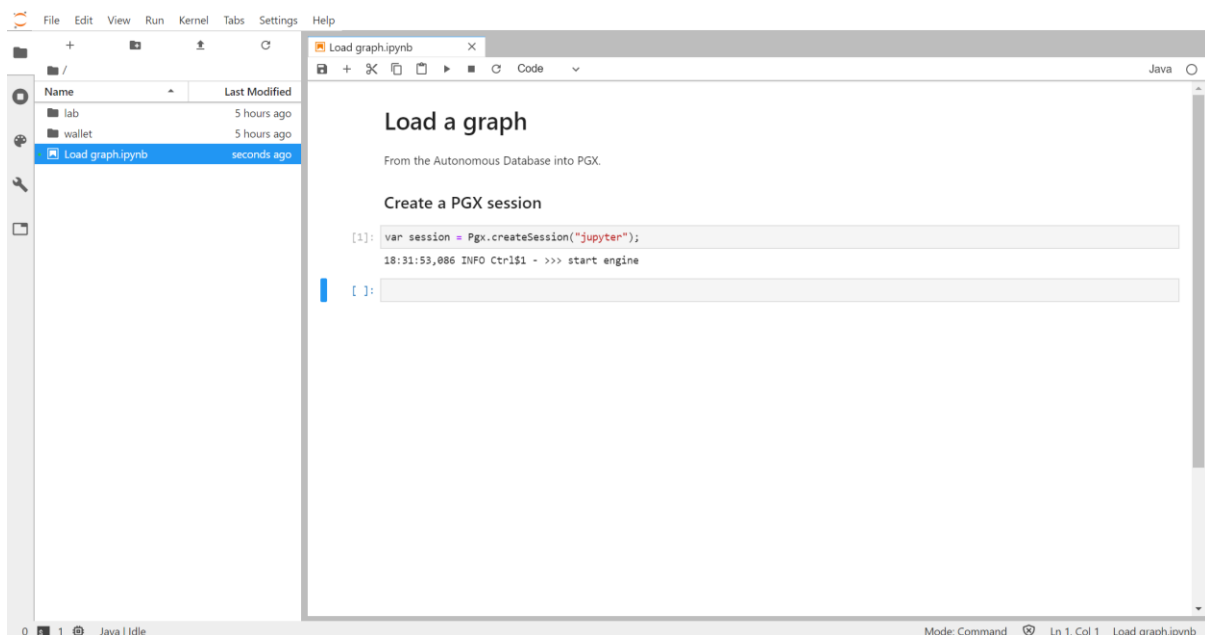


It's a good practice to rename things in a way to describe what they are about. In this case it's highly suggested you give a name to the notebook instead of "Untitled".



## 2) Create a PGX session

```
var session = Pgx.createSession("jupyter");
```





In the PGX client JShell session this variable is already initialized by default when the client session starts. The parameter to the function can be any kind of text, it's a way to identify sessions but in this case, a single user environment, it doesn't matter.

## 2) Create a configuration defining which graph to load

```
var cfg = GraphConfigBuilder.forPropertyGraphRdbms()
.setUsername("database_username")
.setPassword("database_password")
.setName("name_of_the_graph")
.setKeystoreAlias("")
.setJdbcUrl("jdbc:oracle:thin:@connection_descriptor?TNS_ADMIN=path_
to_the_Autonomous_DB_wallet")
.build();
```

The values to use are the following:

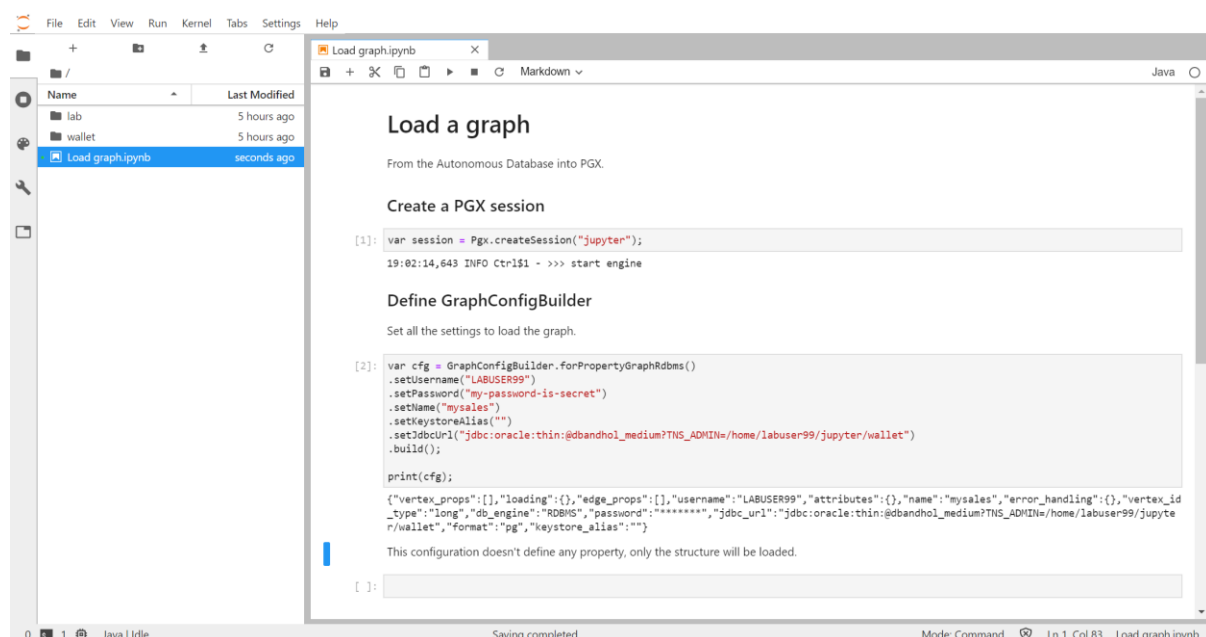
**database\_username**: your username from the SQL Developer Web credentials

**database\_password**: your password from the SQL Developer Web credentials

**name\_of\_the\_graph**: the name you gave to your graph, match the tables names with suffixes "vt\$", "ge\$" etc.

**connection\_descriptor**: database connection descriptor name as defined inside tnsnames.ora, for this lab the value is **dbandhol\_medium**

**path\_to\_the\_Autonomous\_DB\_wallet**: path to the folder where the wallet you downloaded from the Autonomous Database has been extracted. For this lab the value is: **/home/<your JupyterLab username>/jupyter/wallet**. You will need to replace the <your JupyterLab username> by the value from the JupyterLab credentials.





Building this GraphConfigBuilder return the JSON representing the configuration.

### 3) Load the graph

```
var graph1 = session.readGraphWithProperties(cfg);
```

```
[2]: var cfg = GraphConfigBuilder.forPropertyGraphRdbms()
    .setUsername("LABUSER00")
    .setPassword("HOI_gianni99#")
    .setName("mysales")
    .setKeystoreAlias("")
    .setJdbcUrl("jdbc:oracle:thin:@dbandho1_medium?TNS_ADMIN=/home/labuser99/jupyter/wallet")
    .build();

    print(cfg);

    {"keystore_alias":"","username":"LABUSER00","db_engine":"RDBMS","vertex_props":[],"password":"*****","error_handling":{"edge_prop
    s":{"vertex_id_type":"long","loading":{"format":"pg","attributes":{"name":"mysales","jdbc_url":"jdbc:oracle:thin:@dbandho1_mediu
    m?TNS_ADMIN=/home/labuser99/jupyter/wallet"}
    This configuration doesn't define any property, only the structure will be loaded.

    Load the graph

    [3]: var graph1 = session.readGraphWithProperties(cfg);

    print(graph1);

    PgxGraph[name=mysales,N=55595,E=405960,created=1582484648695]

    [4]: print("graph has "+ graph1.getNumVertices() +" vertices, and "+ graph1.getNumEdges() +" edges");

    graph has 55595 vertices, and 405960 edges

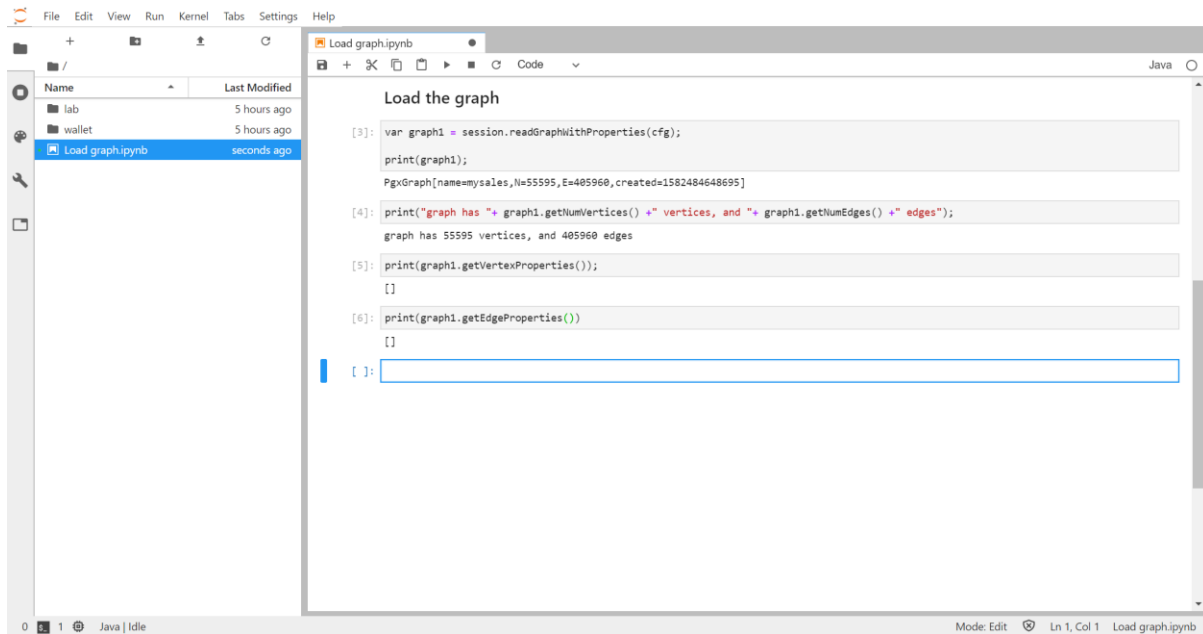
    [ ]:
```

Printing the graph1 variable will return the graph name, the number of edges, the number of nodes and the timestamp when the graph was created.

### 4) Test the graph is really available

```
print("The graph has "+graph1.getNumVertices()+" nodes and
      "+graph1.getNumEdges()+" edges.");
```

```
print(graph1.getVertexProperties());
print(graph1.getEdgeProperties());
```



When you list the properties for nodes and edges there is nothing returned. This is because the loaded graph was just the structure of nodes and edges, but not a single property has been loaded.

5) Load a graph with all the properties and the labels

```

var cfg = GraphConfigBuilder.forPropertyGraphRdbms()
  .setUsername("database_username")
  .setPassword("database_password")
  .setName("name_of_the_graph")
  .setKeystoreAlias("")
  .setJdbcUrl("jdbc:oracle:thin:@connection_descriptor?TNS_ADMIN=path_
to_the_Autonomous_DB_wallet")
  .setLoadEdgeLabel(true)
  .setLoadVertexLabels(true)
  ... add all the properties ...
  .build();
  
```



```

Define GraphConfigBuilder
Set all the settings to load the graph.

[2]: var cfg = GraphConfigBuilder.forPropertyGraphDatabases()
    .setUsername("LABUSER99")
    .setPassword("my-password-is-secret")
    .setName("mysales")
    .setKeystoreAlias("")
    .setJdbcUrl("jdbc:oracle:thin:@dbandho1_medium?TNS_ADMIN=/home/labuser99/jupyter/wallet")
    .setLoadEdgeLabel(true)
    .setLoadVertexLabels(true)
    .setUseVertexPropertyValueAsLabel("label")
    .setDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSSSSXXX")
    .addEdgeProperty("amount", PropertyType.FLOAT)
    .addEdgeProperty("orderDate", PropertyType.LOCAL_DATE)
    .addEdgeProperty("quantity", PropertyType.FLOAT)
    .addEdgeProperty("stateProvince", PropertyType.STRING)
    .addVertexProperty("category", PropertyType.STRING)
    .addVertexProperty("gender", PropertyType.STRING)
    .addVertexProperty("isoCode", PropertyType.STRING)
    .addVertexProperty("listPrice", PropertyType.FLOAT)
    .addVertexProperty("maritalStatus", PropertyType.STRING)
    .addVertexProperty("name", PropertyType.STRING)
    .addVertexProperty("sourceId", PropertyType.INTEGER)
    .addVertexProperty("subcategory", PropertyType.STRING)
    .addVertexProperty("yearOfBirth", PropertyType.INTEGER)
    .build();

    print(cfg);

{"jdbc_url":"jdbc:oracle:thin:@dbandho1_medium?TNS_ADMIN=/home/labuser99/jupyter/wallet","edge_props":[{"name":"amount","type":"float","dimension":0}, {"name":"orderDate","type":"local_date","dimension":0}, {"name":"quantity","type":"float","dimension":0}, {"name":"stateProvince","type":"string","dimension":0}], "vertex_id_type":"long","password":"*****","vertex_props":[{"name":"category","type":"string","dimension":0}, {"name":"gender","type":"string","dimension":0}, {"name":"isoCode","type":"string","dimension":0}, {"name":"listPrice","type":"float","dimension":0}, {"name":"maritalStatus","type":"string","dimension":0}, {"name":"name","type":"string","dimension":0}, {"name":"sourceId","type":"integer","dimension":0}, {"name":"subcategory","type":"string","dimension":0}, {"name":"yearOfBirth","type":"integer","dimension":0}], "date_format":"yyyy-MM-dd'T'HH:mm:ss.SSSSSXXX","username":"LABUSER99","error_handling":{},"attributes":{}
  
```

The list of all the properties can be generated by a query in the database using a query like the one below. A subset of properties can be loaded, it all depends on what are your needs. More properties = more resources = more time to load.

```

WITH properties AS (
    SELECT DISTINCT k, t, 'Vertex' AS kind FROM mysalesvt$
    UNION ALL
    SELECT DISTINCT k, t, 'Edge' AS kind FROM mysalesge$
), cfg AS (
    SELECT '.add' || kind || 'Property("' || k || "',PropertyType.'
    ||
    CASE
        WHEN t = 1 THEN 'STRING' WHEN t = 2 THEN 'INTEGER' WHEN t =
3 THEN 'FLOAT'
        WHEN t = 4 THEN 'DOUBLE' WHEN t = 7 THEN 'LONG' WHEN t = 5
THEN 'LOCAL_DATE'
        WHEN t = 6 THEN 'BOOLEAN'
    END || ')' AS prop
    FROM properties
    WHERE k IS NOT NULL
) SELECT LISTAGG(prop,')' WITHIN GROUP(ORDER BY prop) FROM cfg;
  
```



Oracle SQL Developer interface showing a SQL query in the Worksheet tab. The query is as follows:

```

1 WITH properties AS (
2   SELECT DISTINCT k, t, 'Vertex' AS kind FROM mysalesvt$
3   UNION ALL
4   SELECT DISTINCT k, t, 'Edge' AS kind FROM mysaleses$
5 ), cfg AS (
6   SELECT .add || kind || 'Property(' || k || ', ' || PropertyType || ' ' ||
7     CASE
8       WHEN t = 1 THEN 'STRING' WHEN t = 2 THEN 'INTEGER' WHEN t = 3 THEN 'FLOAT'
9       WHEN t = 4 THEN 'DOUBLE' WHEN t = 7 THEN 'LONG' WHEN t = 5 THEN 'LOCAL_DATE'
10      WHEN t = 6 THEN 'BOOLEAN'
11     END || ')' AS prop
12 FROM properties
13 WHERE k IS NOT NULL
14 ) SELECT LISTAGG(prop, '') WITHIN GROUP (ORDER BY prop) FROM cfg;

```

The Query Result tab shows the output of the query, including a list of properties and a graph structure. A red arrow points to the 'addEdgeProperty' button in the Query Result tab.

## 6) Test the graph is really available

Jupyter Notebook interface showing a Python script that loads a graph with properties. The script is as follows:

```

[3]: var graph2 = session.readGraphWithProperties(cfg);
    print(graph2);

PgxGraph[name=mysales,N=55595,E=405960,created=158248594222]

[4]: print("graph has " + graph2.getNumVertices() + " vertices, and " + graph2.getNumEdges() + " edges");
    graph has 55595 vertices, and 405960 edges

[ ]: print(graph2.getVertexProperties());

[ ]: print(graph2.getEdgeProperties());

[ ]:

```

The output of the script shows the graph structure and the number of vertices and edges. The graph has 55595 vertices and 405960 edges.





```

0, "type": "string", {"name": "isoCode", "dimension": 0, "type": "string"}, {"name": "listPrice", "dimension": 0, "type": "float"}, {"name": "maritalStatus", "dimension": 0, "type": "string"}, {"name": "name", "dimension": 0, "type": "string"}, {"name": "sourceId", "dimension": 0, "type": "integer"}, {"name": "subcategory", "dimension": 0, "type": "string"}, {"name": "yearOfBirth", "dimension": 0, "type": "integer"}, {"name": "password": "*****", "error_handling": {}}
This configuration doesn't define any property, only the structure will be loaded.

Load the graph

[3]: var graph2 = session.readGraphWithProperties(cfg);
print(graph2);
PgxGraph[name=mysales, N=55595, E=405960, created=1582485304222]

[4]: print("graph has "+ graph2.getNumVertices() + " vertices, and "+ graph2.getNumEdges() + " edges");
graph has 55595 vertices, and 405960 edges

[5]: print(graph2.getVertexProperties());
[VertexProperty[name=yearOfBirth, type=integer, graph=mysales], VertexProperty[name=listPrice, type=float, graph=mysales], VertexProperty[name=category, type=string, graph=mysales], VertexProperty[name=maritalStatus, type=string, graph=mysales], VertexProperty[name=subcategory, type=string, graph=mysales], VertexProperty[name=gender, type=string, graph=mysales], VertexProperty[name=name, type=string, graph=mysales], VertexProperty[name=isoCode, type=string, graph=mysales], VertexProperty[name=sourceId, type=integer, graph=mysales]]

[6]: print(graph2.getEdgeProperties());
[EdgeProperty[name=orderDate, type=local_date, graph=mysales], EdgeProperty[name=amount, type=float, graph=mysales], EdgeProperty[name=stateProvince, type=string, graph=mysales], EdgeProperty[name=quantity, type=float, graph=mysales]]

[ ]:

```

This time the graph return a list of properties for both nodes and edges. You can see that this list perfectly matches with all the properties defined in the configuration before to load the graph.

## 7) Query the graph

```
var query = "SELECT c.name, p.name, b.orderDate, b.amount,
b.quantity WHERE (c:customer) -[b:buys]-> (p:product) LIMIT 10";
print(query);
```

```
var resultSet = graph2.queryPgql(query);
print(resultSet);
```

```
for (var result : resultSet) {
    print(result);
}
```



The results of the query are objects referencing all the components of each row.

```
for (var result : resultSet) {
    print(result.getString(1) + " bought " + result.getFloat(5)
        + " " + result.getString(2) + " the " + result.getDate(3) + "
for " + result.getFloat(4) + "$");
}
```

```
for (var result : resultSet) {
    print(result.getString("c.name") + " bought " +
result.getFloat("b.quantity")
        + " " + result.getString("p.name") + " the " +
result.getDate("b.orderDate") + " for " + result.getFloat("b.amount")
+ "$");
}
```



```

oracle.pgx.api.ResultImpl@19750565
oracle.pgx.api.ResultImpl@5711373a
oracle.pgx.api.ResultImpl@5d2ca975
oracle.pgx.api.ResultImpl@2ffe9428
oracle.pgx.api.ResultImpl@2d24a4c9
oracle.pgx.api.ResultImpl@7352eeb0

[11]: for (var result : resultSet) {
      print(result.getString(1) + " bought " + result.getFloat(5)
            + " " + result.getString(2) + " the " + result.getDate(3) + " for " + result.getFloat(4) + "$");
    }

Harriett Charles bought 3.0 Y Box the 2001-06-26 for 898.42$
Holly Kindrid bought 3.0 Y Box the 2000-10-23 for 922.41$
Holly Kindrid bought 1.0 Y Box the 2000-12-23 for 289.02$
Roxana Dodds bought 2.0 Y Box the 2000-05-16 for 611.94$
Roxana Dodds bought 1.0 Y Box the 2000-08-16 for 307.47$
Roxana Dodds bought 1.0 Y Box the 2000-07-16 for 307.47$
Roxana Dodds bought 1.0 Y Box the 2000-03-16 for 301.74$
Ingrid Shore bought 2.0 Y Box the 2001-02-20 for 593.31$
Rita Dobson bought 3.0 Y Box the 2001-02-16 for 900.61$
Rita Dobson bought 2.0 Y Box the 2001-04-17 for 609.69$

[13]: for (var result : resultSet) {
      print(result.getString("c.name") + " bought " + result.getFloat("b.quantity")
            + " " + result.getString("p.name") + " the " + result.getDate("b.orderDate") + " for " + result.getFloat("b.amount") + "$");
    }

Harriett Charles bought 3.0 Y Box the 2001-06-26 for 898.42$
Holly Kindrid bought 3.0 Y Box the 2000-10-23 for 922.41$
Holly Kindrid bought 1.0 Y Box the 2000-12-23 for 289.02$
Roxana Dodds bought 2.0 Y Box the 2000-05-16 for 611.94$
Roxana Dodds bought 1.0 Y Box the 2000-08-16 for 307.47$
Roxana Dodds bought 1.0 Y Box the 2000-07-16 for 307.47$
Roxana Dodds bought 1.0 Y Box the 2000-03-16 for 301.74$
Ingrid Shore bought 2.0 Y Box the 2001-02-20 for 593.31$
Rita Dobson bought 3.0 Y Box the 2001-02-16 for 900.61$
Rita Dobson bought 2.0 Y Box the 2001-04-17 for 609.69$
  
```

The components of each row can be retrieved based on their type (by using `getString(...)`, `getFloat(...)` etc.) and referenced either using a position reference of the column in the result (starting with 1 for the first one) or by name. If an alias has been used for a column this name must be the alias.

## 8) Execute an algorithm on the graph

```
var analyst = session.createAnalyst();
```

```
var pagerank = analyst.pagerank(graph2);
print(pagerank);
```

```
var query = "SELECT v, v.name, v."+pagerank.getName()+" WHERE
(v:product) ORDER BY v."+pagerank.getName()+" DESC LIMIT 10";
print(query);
var resultSet = graph2.queryPgql(query);
for (var result : resultSet) {
    print("node: "+result.getString(2)+" has pagerank =
"+result.getDouble(3));
}
resultSet.close();
```



**Calculate PageRank**

Get the most bought products by calculating the pagerank on the graph.

```
[24]: var analyst = session.createAnalyst();
[25]: var pagerank = analyst.pagerank(graph2);
print(pagerank);
VertexProperty[name=pagerank_2,type=double,graphmysales]
[26]: var query = "SELECT v, v.name, v."+pagerank.getName()+" WHERE (v:product) ORDER BY v."+pagerank.getName()+" DESC LIMIT 10";
print(query);
SELECT v, v.name, v.pagerank_2 WHERE (v:product) ORDER BY v.pagerank_2 DESC LIMIT 10
[27]: var resultSet = graph2.queryPsql(query);
for (var result : resultSet) {
    print("node: "+result.getString(2)+" has pagerank = "+result.getDouble(3));
}
resultSet.close();
```

node: Mouse Pad has pagerank = 4.171326202907747E-4  
node: Keyboard Wrist Rest has pagerank = 3.908997368917471E-4  
node: PCMCIA modem/fax 19200 baud has pagerank = 3.568166138635059E-4  
node: Envoy Ambassador has pagerank = 2.8479815034604367E-4  
node: O/S Documentation Set - English has pagerank = 2.846594888909215E-4  
node: DVD-R Disc with Jewel Case, 4.7 GB has pagerank = 2.544536944631752E-4  
node: PCMCIA modem/fax 28800 baud has pagerank = 2.5096879757561434E-4  
node: Model K8822S Cordless Phone Battery has pagerank = 2.443376717202824E-4  
node: 1.44MB External 3.5" Diskette has pagerank = 2.309978138437909E-4  
node: External 101-key keyboard has pagerank = 2.2507344594601328E-4

To execute algorithms, you first need an object "analyst" which can be obtained from the PGX session.

The result of an algorithm is often a property (or a set of properties) which will be used to store the result of the algorithm on the various nodes and/or edges impacted. This result is useful to know the name of the property holding the result.

A "pagerank" calculation across the graph and retrieving the products with the highest pagerank must, intuitively, be close to the list of products with the highest quantity sold.

**Oracle SQL Developer**

Worksheet: [Worksheet] | LABUSER00

```
1 SELECT p.prod_name, SUM(s.quantity_sold) FROM sh.sales s, sh.products p
2 WHERE s.prod_id = p.prod_id
3 AND s.time_id >= to_date('20000101', 'yyyymmdd')
4 GROUP BY p.prod_name
5 ORDER BY 2 DESC
6 FETCH FIRST 10 ROWS ONLY;
```

**Query Result**

	prod_name	sum(s.quantity_sold)
1	Mouse Pad	15541
2	Keyboard Wrist Rest	14882
3	PCMCIA modem/fax 19200 baud	14341
4	DVD-R Discs, 4.7GB, Pack of 5	11425
5	PCMCIA modem/fax 28800 baud	11355
6	DVD-R Disc with Jewel Case, 4.7 GB	11329
7	O/S Documentation Set - English	11175
8	CD-R with Jewel Cases, pACK OF 12	11142
9	External 101-key keyboard	10984
10	Model K8822S Cordless Phone Battery	10468

1 | 0 | 0 | 7:41:36 PM - 10 rows total



You can see that the “pagerank” TOP10 calculation on products has many common elements with the TOP10 of the products by quantity sold retrieved by a query in the database.



Slide 56

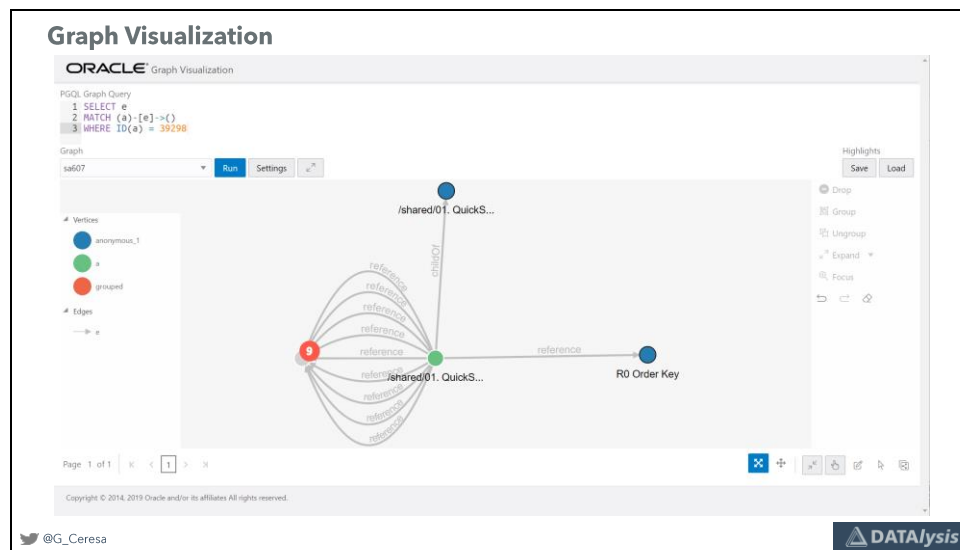
**Graph Visualization, Cytoscape, custom visualizations, REST interface,  
PGQL to SQL translation, ML**

 @G\_Ceresa

 DATAlysis



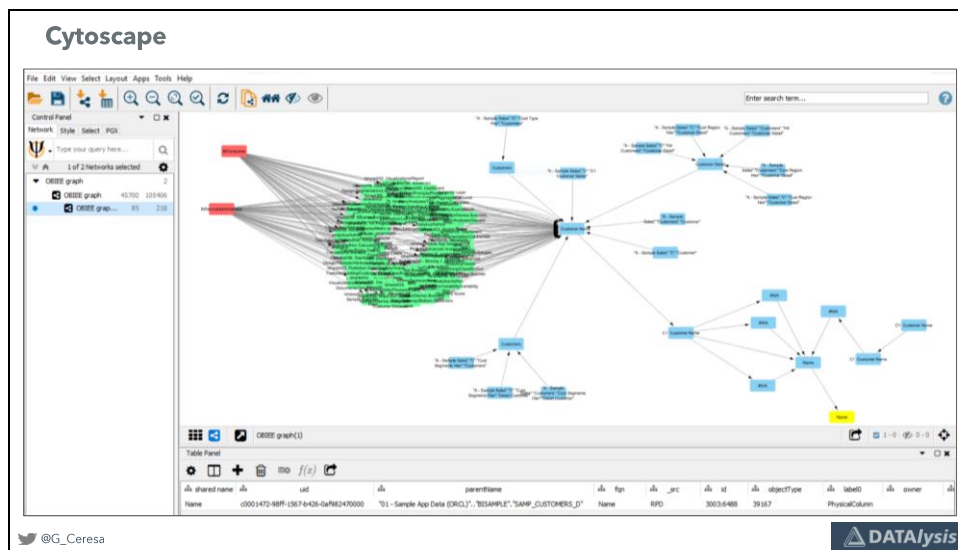
## Slide 57



If you use Graph server 20.1.0 standalone it will by default allow you to access Graph Visualization. A web interface in which you can enter a PGQL query and navigate from there through the results.



Slide 58

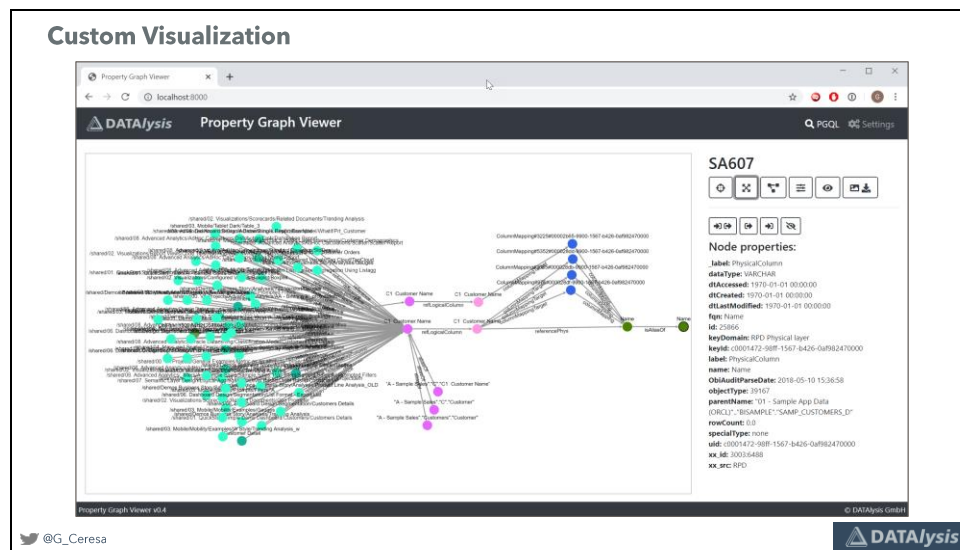


Another visual alternative is using Cytoscape. An open source application you can install and for which Oracle provides a plugin to add support for PGX with graphs stored in the database.





Slide 59



Thanks to the REST web service you can also develop your own visualization tool or integration. Using various libraries like for example CytoscapeJS to easily visualize and interact with a graph in a web application.

Slide 60

# REST interface

The REST web service is fully documented, a swagger.json file is provided.



Slide 61

### PGQL to SQL translation

#### 4.8 Executing PGQL Queries Directly Against Oracle Database

This topic explains how you can execute PGQL queries directly against the graph in Oracle Database (as opposed to in-memory).

Property Graph Query Language (PGQL) queries can be executed against disk-resident property graph data stored in Oracle Database. PGQL on Oracle Database (RDBMS) provides a Java API for executing PGQL queries. Logic in PGQL on RDBMS translates a submitted PGQL query into an equivalent SQL query, and the resulting SQL is executed on the database server. PGQL on RDBMS then wraps the SQL query results with a convenient PGQL result set API.

This PGQL query execution flow is shown in the following figure.

**Figure 4-2 PGQL on Oracle Database (RDBMS)**

The basic execution flow is:

1. The PGQL query is submitted to PGQL on RDBMS through a Java API.
2. The PGQL query is translated to SQL.
3. The translated SQL is submitted to Oracle Database by JDBC.
4. The SQL result set is wrapped as a PGQL result set and returned to the caller.

The ability to execute PGQL queries directly against property graph data stored in Oracle Database provides several benefits.

- PGQL provides a more natural way to express graph queries than SQL manually

@G\_Ceresa

DATAlysis

If you want to translate a PGQL query into SQL to execute it directly in the database bypassing PGX (which means you don't have to load the graph into PGX but you can use directly in the database), you can find the details in the documentation.



Slide 62

**Machine Learning**

On top of the graph algorithms, which are a kind of machine learning already, there is also "real" machine learning which is possible with Oracle Property Graphs.

You can find out more by having a look at this talk I did on the topic:

<https://speakerdeck.com/gianniceresa/when-machine-learning-meets-graph-databases-6eebb419-fa32-4e7e-af03-aafaf601f759>

Right now this part isn't available in the Graph Server 20.1.0, it was a beta feature in the Oracle Labs release of PGX. It's supposed to be available in the product later this year.

*(I have no influence on Oracle plans and strategy, therefore things could still change)*

 @G\_Ceresa



Machine Learning is already possible with graphs, and more advanced algorithms are supposed to be available in the product soon.



Slide 63

**Next steps and conclusion**

 @G\_Ceresa

 DATAlysis



Slide 64

**Next steps**

- Setup an Autonomous Database using the Always-free tier
- Download the Graph 20.1.0 client on your laptop
- Use the JShell client in "almost" the same way you used the notebook

Property Graphs for free with up to 20Gb of storage in the database

@G\_Ceresa

You can easily get started practicing and experimenting Oracle property graph by using the Always-free tier in the Oracle cloud. Create an account and setup your Autonomous Data warehouse instance. Create your graphs there and use the PGX client from your own laptop to directly connect to that database and perform queries or algorithms on the graph.