



Concepts for Specialised Databases

Graph Databases: Neo4j

Objectives

As Yogi Berra said, "In theory there's no difference between theory and practice. But in practice, there is." This statement is especially true for NOSQL databases.

This is a hands-on lab to get familiar with Graph Databases. The main goals are as follows. Given a relational database schema and a query workload, you are supposed to:

- Decide how to design the corresponding graph database,
- Query the database created in order to answer each of the queries.

Try not to think relational-wise and also grasp the subtle details. That is, get familiar with the pros and cons of such systems so that in the future you can decide when to use them.

Lab organization

On session 1, the lecturer will be present at the lab and will help you out. Thus, it is a session to solve your doubts and help you to prepare the final delivery.

On session 2, you must upload the deliverables specified in the corresponding section of this document.

Your lab mate for this practice will be that of the corresponding team creation event.

Important: You are highly advised to attend the lab session.

Tools

This lab requires Neo4j. We assume you can install Neo4j in your own laptop. If this is not possible, contact us.

Training (Activities to do beforehand)

- First, set Neo4j up in your laptop. There are plenty of manuals with lots of information about this, for example, this is the most basic one: <https://neo4j.com/docs/desktop-manual/current/>
- Next, decide what API you want to use: <https://neo4j.com/docs/getting-started/languages-guides/>
- Get familiar with the basics. Create a project and include Neo4j in it. Try to insert some nodes and edges.

Anyway, this training is up to you; decide which manuals suit better your needs. Maybe some other manuals help you better (a comprehensive list of the official manuals can be found here: <https://neo4j.com/docs/>). You are expected to need a couple of hours to set the environment and get familiar with the basics, but this time really depends on each of you.

Session statement

Once you are familiar with Neo4j, solve the following exercise.

For this lab we will consider the TPC-H benchmark (<http://www.tpc.org/tpch/>). The schema of the TPC-H benchmark is introduced in Appendix A. You must design this database in Neo4j (i.e., model it as a graph) in such a way the performance of the following queries is optimal.



Note that the workload is expressed in SQL but you will need to adapt it to Neo4j once you have designed your database.

To implement these queries, you may want to consider indexes or add extra edges to allow navigating the graph in an easier way.

Query 1

```
SELECT l_returnflag, l_linestatus, sum(l_quantity) as sum_qty,  
       sum(l_extendedprice) as sum_base_price,  
       sum(l_extendedprice*(1-l_discount)) as sum_disc_price,  
       sum(l_extendedprice*(1-l_discount)*(1+l_tax)) as sum_charge,  
       avg(l_quantity) as avg_qty, avg(l_extendedprice) as avg_price,  
       avg(l_discount) as avg_disc, count(*) as count_order  
FROM lineitem  
WHERE l_shipdate <= '[date]'  
GROUP BY l_returnflag, l_linestatus  
ORDER BY l_returnflag, l_linestatus;
```

Where [date] is a constant that may vary between executions of the query.

Query 2

```
SELECT s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone,  
       s_comment  
FROM part, supplier, partsupp, nation, region  
WHERE p_partkey = ps_partkey AND s_suppkey = ps_suppkey AND p_size = [SIZE]  
      AND p_type like '%[TYPE] AND s_nationkey = n_nationkey  
      AND n_regionkey = r_regionkey AND r_name = '[REGION]'  
      AND ps_supplycost =  
          (SELECT min(ps_supplycost)  
           FROM partsupp, supplier, nation, region WHERE p_partkey = ps_partkey  
            AND s_suppkey = ps_suppkey AND s_nationkey = n_nationkey  
            AND n_regionkey = r_regionkey AND r_name = '[REGION]')  
ORDER BY s_acctbal desc, n_name, s_name, p_partkey;
```

Where [size], [type] and [region] are constants that may vary between executions of the query.

Query 3

```
SELECT l_orderkey, sum(l_extendedprice*(1-l_discount)) as revenue,  
       o_orderdate, o_shippriority  
FROM customer, orders, lineitem  
WHERE c_mktsegment = '[SEGMENT]' AND c_custkey = o_custkey  
      AND l_orderkey = o_orderkey AND o_orderdate < '[DATE1]'  
      AND l_shipdate > '[DATE2]'  
GROUP BY l_orderkey, o_orderdate, o_shippriority  
ORDER BY revenue desc, o_orderdate;
```

Where [segment], [date1] and [date2] are constants that may vary between executions of the query.



Query 4

```
SELECT n_name, sum(l_extendedprice * (1 - l_discount)) as revenue
FROM customer, orders, lineitem, supplier, nation, region
WHERE c_custkey = o_custkey AND l_orderkey = o_orderkey
      AND l_suppkey = s_suppkey AND c_nationkey = s_nationkey
      AND s_nationkey = n_nationkey AND n_regionkey = r_regionkey
      AND r_name = '[REGION]' AND o_orderdate >= date '[DATE]'
      AND o_orderdate < date '[DATE]' + interval '1' year
GROUP BY n_name
ORDER BY revenue desc;
```

Where [date] and [region] are constants that may vary between executions of the query.

Submission

You need to deliver the following:

- **A PDF explanation document (3 pages long)**, including:
 - The chosen structure for your Neo4j database (showing nodes, edges, and node/edge properties). You should clearly (quantitatively) justify your decisions, paying special attention to nodes/edges created to speed up the queries, and to indexes (if any).
 - The definition in Cypher (<https://neo4j.com/docs/cypher-manual/current/>) of the 4 queries introduced in Appendix B, according to the Neo4j database you designed.
- **A program (in Java, Python, etc.) with your solution**. Specifically, this file must contain:
 - Some inserts meeting the structure described in your explanation document.
 - (Optionally) Your code to implement and execute the 4 queries introduced in Appendix B.

Then upload a .zip file containing both files into the corresponding activity in LearnSQL before the deadline.



Appendix A

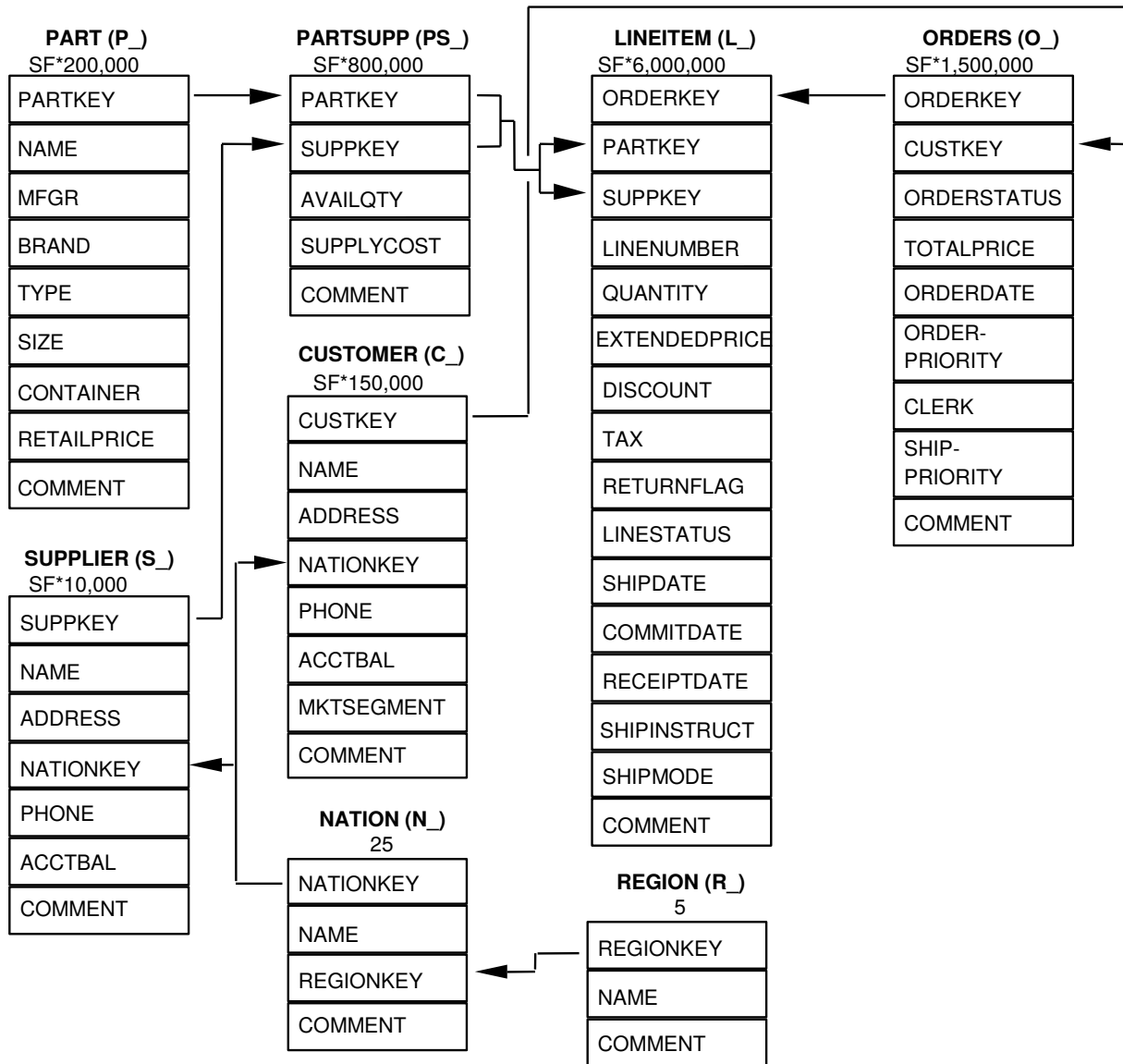
In this appendix you will find a graphical representation of the TPC-H schema and further explanations on the schema you may want to check.

IMPORTANT NOTE: This appendix introduces you to the TPC-H benchmark. This is just for your information (for better understanding each attribute and table), so do not consider the additional information about insertion rules, etc. you may find there.

1.2 Database Entities, Relationships, and Characteristics

The components of the TPC-H database are defined to consist of eight separate and individual tables (the Base Tables). The relationships between columns of these tables are illustrated in Figure 2: The TPC-H Schema.

Figure 2: The TPC-H Schema



Legend:

- The parentheses following each table name contain the prefix of the column names for that table;
- The arrows point in the direction of the one-to-many relationships between tables;
- The number/formula below each table name represents the cardinality (number of rows) of the table. Some are factored by SF, the Scale Factor, to obtain the chosen database size. The cardinality for the LINEITEM table is approximate (see Clause 4.2.5).

1.3 Datatype Definitions

1.3.1 The following datatype definitions apply to the list of columns of each table:

- **Identifier** means that the column must be able to hold any key value generated for that column and be able to support at least 2,147,483,647 unique values;

Comment: A common implementation of this datatype will be an integer. However, for SF greater than 300 some column values will exceed the range of integer values supported by a 4-byte integer. A test sponsor may use some other datatype such as 8-byte integer, decimal or character string to implement the identifier datatype;

- **Integer** means that the column must be able to exactly represent integer values (i.e., values in increments of 1) in the range of at least -2,147,483,646 to 2,147,483,647.
- **Decimal** means that the column must be able to represent values in the range -9,999,999,999.99 to +9,999,999,999.99 in increments of 0.01; the values can be either represented exactly or interpreted to be in this range;
- **Big Decimal** is of the Decimal datatype as defined above, with the additional property that it must be large enough to represent the aggregated values stored in temporary tables created within query variants;
- **Fixed text, size N** means that the column must be able to hold any string of characters of a fixed length of N.

Comment: If the string it holds is shorter than N characters, then trailing spaces must be stored in the database or the database must automatically pad with spaces upon retrieval such that a CHAR_LENGTH() function will return N.

- **Variable text, size N** means that the column must be able to hold any string of characters of a variable length with a maximum length of N. Columns defined as "variable text, size N" may optionally be implemented as "fixed text, size N";
- **Date** is a value whose external representation can be expressed as YYYY-MM-DD, where all characters are numeric. A date must be able to express any day within at least 14 consecutive years. There is no requirement specific to the internal representation of a date.

Comment: The implementation datatype chosen by the test sponsor for a particular datatype definition must be applied consistently to all the instances of that datatype definition in the schema, except for identifier columns, whose datatype may be selected to satisfy database scaling requirements.

1.3.2 The symbol SF is used in this document to represent the scale factor for the database (see Clause 4:).

1.4 Table Layouts

1.4.1 Required Tables

The following list defines the required structure (list of columns) of each table.

The annotations 'Primary Key' and 'Foreign Key', as used in this Clause, are for information only and do not imply additional requirements to implement **primary key** and **foreign key** constraints (see Clause 1.4.2).

PART Table Layout

<u>Column Name</u>	<u>Datatype Requirements</u>	<u>Comment</u>
P_PARTKEY	identifier	SF*200,000 are populated
P_NAME	variable text, size 55	
P_MFGR	fixed text, size 25	

P_BRAND	fixed text, size 10
P_TYPE	variable text, size 25
P_SIZE	integer
P_CONTAINER	fixed text, size 10
P_RETAILPRICE	decimal
P_COMMENT	variable text, size 23
Primary Key: P_PARTKEY	

SUPPLIER Table Layout

<u>Column Name</u>	<u>Datatype Requirements</u>	<u>Comment</u>
S_SUPPKEY	identifier	SF*10,000 are populated
S_NAME	fixed text, size 25	
S_ADDRESS	variable text, size 40	
S_NATIONKEY	Identifier	Foreign Key to N_NATIONKEY
S_PHONE	fixed text, size 15	
S_ACCTBAL	decimal	
S_COMMENT	variable text, size 101	
Primary Key: S_SUPPKEY		

PARTSUPP Table Layout

<u>Column Name</u>	<u>Datatype Requirements</u>	<u>Comment</u>
PS_PARTKEY	Identifier	Foreign Key to P_PARTKEY
PS_SUPPKEY	Identifier	Foreign Key to S_SUPPKEY
PS_AVAILQTY	integer	
PS_SUPPLYCOST	Decimal	
PS_COMMENT	variable text, size 199	
Primary Key: PS_PARTKEY, PS_SUPPKEY		

CUSTOMER Table Layout

<u>Column Name</u>	<u>Datatype Requirements</u>	<u>Comment</u>
C_CUSTKEY	Identifier	SF*150,000 are populated

C_NAME	variable text, size 25	
C_ADDRESS	variable text, size 40	
C_NATIONKEY	Identifier	Foreign Key to N_NATIONKEY
C_PHONE	fixed text, size 15	
C_ACCTBAL	Decimal	
C_MKTSEGMENT	fixed text, size 10	
C_COMMENT	variable text, size 117	
Primary Key: C_CUSTKEY		

ORDERS Table Layout

<u>Column Name</u>	<u>Datatype Requirements</u>	<u>Comment</u>
O_ORDERKEY	Identifier	SF*1,500,000 are sparsely populated
O_CUSTKEY	Identifier	Foreign Key to C_CUSTKEY
O_ORDERSTATUS	fixed text, size 1	
O_TOTALPRICE	Decimal	
O_ORDERDATE	Date	
O_ORDERPRIORITY	fixed text, size 15	
O_CLERK	fixed text, size 15	
O_SHIPPRIORITY	Integer	
O_COMMENT	variable text, size 79	
Primary Key: O_ORDERKEY		

Comment: Orders are not present for all customers. In fact, one-third of the customers do not have any order in the database. The orders are assigned at random to two-thirds of the customers (see Clause 4:). The purpose of this is to exercise the capabilities of the DBMS to handle "dead data" when joining two or more tables.

LINEITEM Table Layout

<u>Column Name</u>	<u>Datatype Requirements</u>	<u>Comment</u>
L_ORDERKEY	identifier	Foreign Key to O_ORDERKEY
L_PARTKEY	identifier	Foreign key to P_PARTKEY, first part of the compound Foreign Key to (PS_PARTKEY, PS_SUPPKEY) with L_SUPPKEY
L_SUPPKEY	Identifier	Foreign key to S_SUPPKEY, second part of the compound Foreign Key to (PS_PARTKEY,

PS_SUPPKEY) with L_PARTKEY

L_LINENUMBER	integer
L_QUANTITY	decimal
L_EXTENDEDPRICE	decimal
L_DISCOUNT	decimal
L_TAX	decimal
L_RETURNFLAG	fixed text, size 1
L_LINESTATUS	fixed text, size 1
L_SHIPDATE	date
L_COMMITDATE	date
L_RECEIPTDATE	date
L_SHIPINSTRUCT	fixed text, size 25
L_SHIPMODE	fixed text, size 10
L_COMMENT	variable text size 44
Primary Key: L_ORDERKEY, L_LINENUMBER	

NATION Table Layout

<u>Column Name</u>	<u>Datatype Requirements</u>	<u>Comment</u>
N_NATIONKEY	identifier	25 nations are populated
N_NAME	fixed text, size 25	
N_REGIONKEY	identifier	Foreign Key to R_REGIONKEY
N_COMMENT	variable text, size 152	
Primary Key: N_NATIONKEY		

REGION Table Layout

<u>Column Name</u>	<u>Datatype Requirements</u>	<u>Comment</u>
R_REGIONKEY	identifier	5 regions are populated
R_NAME	fixed text, size 25	
R_COMMENT	variable text, size 152	
Primary Key: R_REGIONKEY		