# Introduction to Neo4j

<https://neo4j.com/docs/getting-started/current/>

**Properties** are name-value pairs that are used to add qualities to nodes and relationships.

A **schema** in Neo4j refers to indexes and constraints.

Neo4j is often described as schema optional, meaning that it is not necessary to create indexes and constraints.

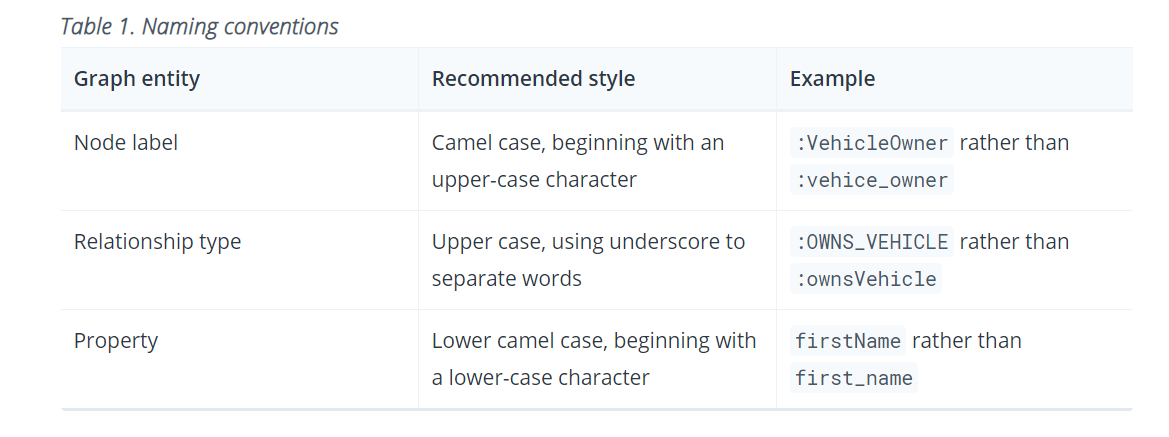
Indexes and constraints can be introduced when desired, in order to gain performance or modeling benefits.

Node labels, relationship types and properties are case sensitive,

Neo4j’s Property Graphs are composed of nodes and relationships, either of which may have properties.

The real strength of the property graph lies in its ability to encode **patterns** of connected nodes and relationships.

In most cases, we want to connect new data to existing structures. This requires that we know how to find existing patterns in our graph data,



**CREATE** (a:Person { name:"Tom Hanks",

born:1956 })-[r:ACTED\_IN { roles: ["Forrest"]}]->(m:Movie { title:"Forrest Gump",released:1994 })

CREATE (d:Person { name:"Robert Zemeckis", born:1951 })-[:DIRECTED]->(m)

RETURN a,d,r,m

**MATCH** (p:Person { name:"Tom Hanks" })-[r:ACTED\_IN]->(m:Movie)

RETURN m.title, r.roles

To extend the graph with new information, we first match the existing connection points and then attach the newly created nodes to them with relationships.

It is important to remember that we can assign variables to both nodes and relationships and use them later on, no matter if they were created or matched.

You can alleviate some of that by creating supporting indexes or constraints, which we’ll discuss later. But it’s still not for free,

For instance, if we return person.name we can still **ORDER BY** person.age since both are accessible from the person reference. **We cannot order by things that are not returned.** This is especially important with aggregation and DISTINCT return values, since both remove the visibility of data that is aggregated.

In Cypher it is possible to chain fragments of statements together, similar to how it is done within a **data-flow pipeline**. Each fragment works on the output from the previous one, and its results can feed into the next one. Only columns declared in the **WITH** clause are available in subsequent query parts.

WITH is similar to the RETURN clause. The difference is that the WITH clause does not finish the query, but prepares the input for the next part.Expressions, aggregations, ordering and pagination can be used in the same way as in the RETURN clause.

**Constraints** are used to make sure that the data adheres to the rules of the domain.

Constraints can be added to database that already has data in it. This requires that the existing data complies with the constraint that is being added.

从csv文件导入数据

<https://neo4j.com/docs/getting-started/current/cypher-intro/load-csv/#_the_data_files>

# Cypher Basic

<https://neo4j.com/docs/cypher-manual/current/introduction/>

**MATCH**: The graph pattern to match. This is the most common way to get data from the graph.

**WHERE**: Not a clause in its own right, but rather part of MATCH, OPTIONAL MATCH and WITH. Adds constraints to a pattern, or filters the intermediate result passing through WITH.

**RETURN**: What to return.

**CREATE** (and **DELETE**): Create (and delete) nodes and relationships.

**SET** (and REMOVE): Set values to properties and add labels on nodes using SET and use REMOVE to remove them.

**MERGE**: Match existing or create new nodes and patterns. This is especially useful together with unique constraints.

**MATCH** (n {name: 'John'})-[:FRIEND]-(friend)

**WITH** n, count(friend) AS friendsCount

**WHERE** friendsCount > 3

**RETURN** n, friendsCount

Using **WITH**, you specify how you want the aggregation to happen, and that the aggregation has to be finished before Cypher can start filtering.

**MATCH** (n {name: 'John'})-[:FRIEND]-(friend)

**WITH** n, count(friend) AS friendsCount

**SET** n.friendsCount = friendsCount

**RETURN** n.friendsCount

You can **chain together** as many query parts as the available memory permits.

Every statement is executed within the context of the transaction, and nothing will be persisted to disk until that transaction is successfully committed.

In short, an updating query will always either fully succeed, or not succeed at all.

When writing procedures or using Neo4j embedded, remember that all **iterators** returned from an execution result should be either fully exhausted or closed. This ensures that the resources bound to them are properly released.

**In Neo4j, all relationships have a direction.** However, you can have the notion of undirected relationships at query time.

Cypher makes use of **relationship isomorphism** for path matching.

# Cypher常用命令

1. 显示所有节点和关系

MATCH (m)—(n) RETURN n

1. 清空数据库

MATCH (n) DETACH DELETE n

1. 导出数据库

<https://neo4j.com/docs/operations-manual/current/backup-restore/offline-backup/#offline-backup>

bin/neo4j-admin dump --database=neo4j --to=/dumps/neo4j/neo4j-<timestamp>.dump

1. 导入数据库

<https://neo4j.com/docs/operations-manual/current/backup-restore/restore-dump/#restore-dump>

bin/neo4j-admin load --from=/dumps/neo4j/neo4j-<timestamp>.dump --database=neo4j –force

注意：新建的数据库名称中不能包含大写字母和下划线

# Python Driver API

<https://neo4j.com/docs/api/python-driver/4.2/api.html>

**Driver Object Lifetime**

For general applications, it is recommended to create one top-level **neo4j.Driver** object that lives for the lifetime of the application.

Connection details held by the neo4j.Driver are **immutable**. Therefore if, for example, a password is changed, a replacement neo4j.Driver object must be created. More than one Driver may be required if connections to multiple databases, or connections as multiple users, are required.

neo4j.Driver objects are **thread-safe** but cannot be shared across processes. Therefore, multithreading should generally be preferred over multiprocessing for parallel database access. If using multiprocessing however, each process will require its own neo4j.Driver object.

**Sessions & Transactions**

All database activity is co-ordinated through two mechanisms: the neo4j.Session and the neo4j.Transaction.

A neo4j.Session is **a logical container** for any number of causally-related transactional units of work. Sessions automatically provide guarantees of causal consistency within a clustered environment but multiple sessions can also be causally chained if required. Sessions provide the top-level of containment for database activity. Session creation is a lightweight operation and sessions are **not thread safe**.

Connections are drawn from the neo4j.Driver connection pool as required.

A neo4j.Transaction is a unit of work that is either committed in its entirety or is rolled back on failure.

Sessions will often be created and destroyed using a **with block context**.

Session creation is a lightweight operation and sessions are not thread safe. **Therefore a session should generally be short-lived, and not span multiple threads.**

transactions --> tx

read\_transaction(transaction\_function, \*args, \*\*kwargs)

Execute a unit of work in a managed **read** transaction.

**transaction\_function**–a function that takes a transaction as an argument and does work with the transaction. tx\_function(tx, \*args, \*\*kwargs)

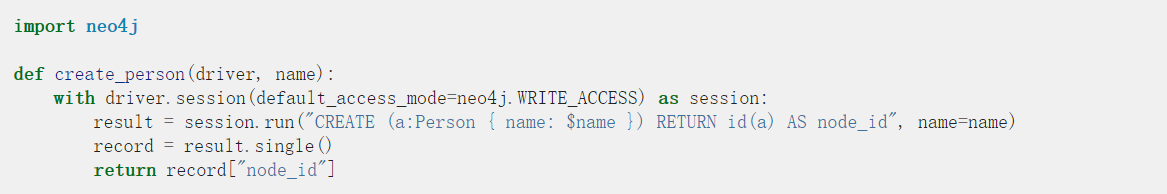
write\_transaction(transaction\_function, \*args, \*\*kwargs)

Execute a unit of work in a managed **write** transaction.

<https://neo4j.com/docs/api/python-driver/4.2/api.html#transaction>

**Auto-commit Transaction**

Auto-commit transactions are the simplest form of transaction, available via neo4j.Session.run().



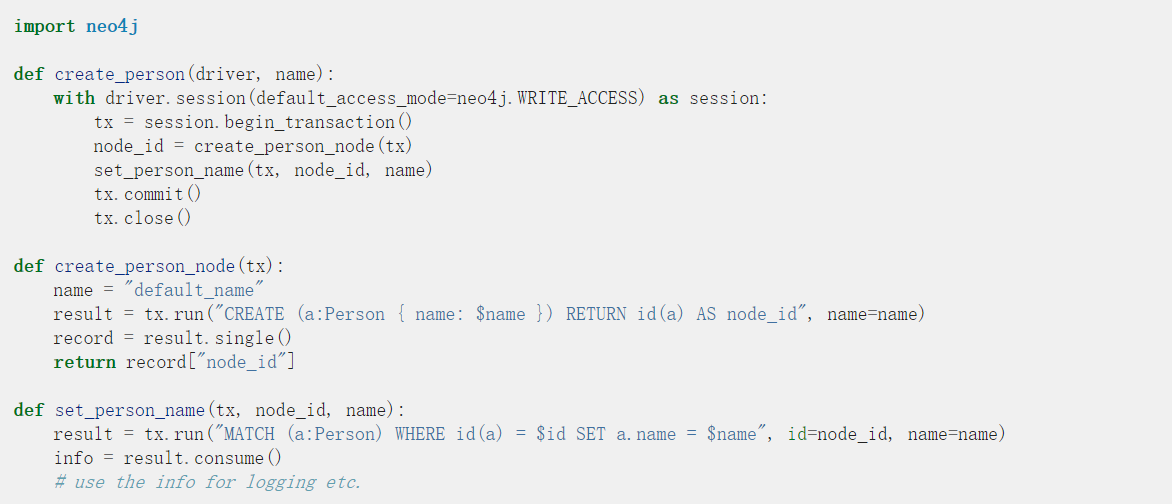
These are easy to use but support only **one statement per transaction** and are **not automatically retried on failure**.

**Explicit Transactions**

Explicit transactions support multiple statements and must be created with an explicit neo4j.Session.begin\_transaction() call.

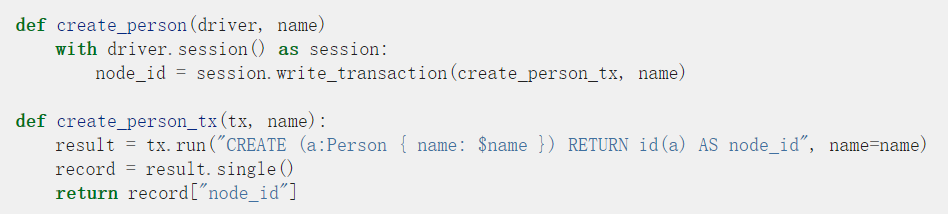
Closing an explicit transaction can either happen automatically at the end of a **with** block, or can be explicitly controlled through the neo4j.Transaction.commit(), neo4j.Transaction.rollback() or neo4j.Transaction.close() methods.

Explicit transactions are most useful for applications that need to distribute Cypher execution across multiple functions for the same transaction.



**Managed Transactions**

These allow a **function object(Anything is obj in Python)** representing the transactional unit of work to be passed as a **parameter**. This function is called one or more times, within a configurable time limit, **until it succeeds**. Results should be fully consumed within the function and **only aggregate or status values should be returned**. **Returning a live result object would prevent the driver from correctly managing connections and would break retry guarantees.**



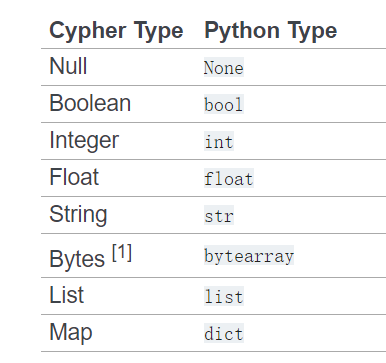
single()

Obtain the next and only remaining record from this result if available else return None. Calling this method always exhausts the result.

peek()

Obtain the next record from this result without consuming it. This leaves the record in the buffer for further processing.

Cypher supports a set of core data types that all map to built-in types in Python.



<https://neo4j.com/docs/api/python-driver/4.2/api.html#graph-data-types>

class neo4j.graph.Graph

A local, self-contained graph object that acts as a container for **Node** and **neo4j.Relationship** instances. This is typically obtained via the **neo4j.Result.graph()** method.

nodes

Access a set view of the nodes in this graph.

relationships

Access a set view of the relationships in this graph.

class neo4j.graph.Node

Self-contained graph node.

keys()

Return an iterable of all property names.

values()

Return an iterable of all property values.

items()

Return an iterable of all property name-value pairs.

labels

The set of labels attached to this node.

class neo4j.graph.Relationship

Self-contained graph relationship.

start\_node

The start node of this relationship.

end\_node

The end node of this relationship.

keys()

Return an iterable of all property names.

values()¶

Return an iterable of all property values.

items()

Return an iterable of all property name-value pairs.

<https://neo4j.com/docs/python-manual/current/client-applications/>

The configuration details are **immutable** for the lifetime of the Driver Object. Therefore, if multiple configurations are required (such as when working with multiple database users) then multiple Driver Objects must be used.

<https://neo4j.com/docs/cypher-manual/current/introduction/quering-updating-administering/>

Any query can return data. If a query only reads, it has to return data. If a read-query doesn’t return any data, it serves no purpose, and is therefore not a valid Cypher query. Queries that update the graph don’t have to return anything, but they can.

If you return nodes or relationships from a query that has just deleted them — beware, you are holding a **pointer** that is no longer valid.

<https://neo4j.com/docs/cypher-manual/current/introduction/transactions/>

All Cypher statements are explicitly run within a transaction. For read-only queries, the transaction will always succeed. For updating queries it is possible that a failure can occur for some reason, for example if the query attempts to violate a constraint, in which case the entire transaction is rolled back, and no changes are made to the graph. **Every statement is executed within the context of the transaction, and nothing will be persisted to disk until that transaction is successfully committed.**

Note that the transaction will hold the changes in memory until the whole query, or whole set of queries, has finished executing. A query that makes a large number of updates will consequently use large amounts of memory.

When writing procedures or using Neo4j embedded, remember that all iterators returned from an execution result should be either fully exhausted or closed. This ensures that the resources bound to them are properly released.