



Cypher is the declarative query language for Neo4j, the world's leading graph database.

Key principles and capabilities of Cypher are as follows:

- Cypher matches patterns of nodes and relationships in the graph, to extract information or modify the data.
- Cypher has the concept of variables which denote named, bound elements and parameters.
- Cypher can create, update, and remove nodes, relationships, labels, and properties.
- Cypher manages indexes and constraints.

You can try Cypher snippets live in the Neo4j Console at [console.neo4j.org](https://console.neo4j.org) or read the full Cypher documentation in the [Neo4j Developer Manual](#). For live graph models using Cypher check out [GraphGist](#).

The Cypher Refcard is also [available in PDF format](#).

Note: {value} denotes either literals or maps, used for ad hoc Cypher queries. The usage of parameters is recommended in applications, and are denoted by \$value. Neo4j properties can be strings, numbers, booleans or arrays thereof. Cypher also supports maps and lists.

Legend

Read
Write
General
Functions
Schema
Performance

Syntax

Read Query Structure
<pre>[MATCH WHERE] [OPTIONAL MATCH WHERE] [WITH [ORDER BY] [SKIP] [LIMIT]] RETURN [ORDER BY] [SKIP] [LIMIT]</pre>

MATCH
<pre>MATCH (n:Person)-[:KNOWS]-&gt;(m:Person) WHERE n.name = 'Alice'</pre> Node patterns can contain labels and properties.
<pre>MATCH (n)--&gt;(m)</pre> Any pattern can be used in MATCH.
<pre>MATCH (n {name: 'Alice'})--&gt;(m)</pre> Patterns with node properties.
<pre>MATCH p = (n)--&gt;(m)</pre> Assign a path to p.
<pre>OPTIONAL MATCH (n)-[r]-&gt;(m)</pre> Optional pattern: nulls will be used for missing parts.

WHERE
<pre>WHERE n.property &lt;&gt; \$value</pre> Use a predicate to filter. Note that WHERE is always part of a MATCH, OPTIONAL MATCH, WITH or START clause. Putting it after a different clause in a query will alter what it does.

Write-Only Query Structure
<pre>(CREATE [UNIQUE]   MERGE)* [SET DELETE REMOVE FOREACH]* [RETURN [ORDER BY] [SKIP] [LIMIT]]</pre>

Read-Write Query Structure
<pre>[MATCH WHERE] [OPTIONAL MATCH WHERE] [WITH [ORDER BY] [SKIP] [LIMIT]] (CREATE [UNIQUE]   MERGE)* [SET DELETE REMOVE FOREACH]* [RETURN [ORDER BY] [SKIP] [LIMIT]]</pre>

CREATE
<pre>CREATE (n {name: \$value})</pre> Create a node with the given properties.
<pre>CREATE (n \$map)</pre> Create a node with the given properties.
<pre>UNWIND \$listOfMaps AS properties CREATE (n) SET n = properties</pre> Create nodes with the given properties.
<pre>CREATE (n)-[r:KNOWS]-&gt;(m)</pre> Create a relationship with the given type and direction; bind a variable to it.
<pre>CREATE (n)-[:LOVES {since: \$value}]-&gt;(m)</pre> Create a relationship with the given type, direction, and properties.

SET
<pre>SET n.property1 = \$value1,     n.property2 = \$value2</pre> Update or create a property.
<pre>SET n = \$map</pre> Set all properties. This will remove any existing properties.
<pre>SET n += \$map</pre> Add and update properties, while keeping existing ones.
<pre>SET n:Person</pre> Adds a label Person to a node.

Import
<pre>LOAD CSV FROM 'https://neo4j.com/docs/cypher-refcard/3.3/csv/artists- refcard/3.3/csv/artists.csv' AS line CREATE (:Artist {name: line[1], year: toInteger(line[2])})</pre> Load data from a CSV file and create nodes.
<pre>LOAD CSV WITH HEADERS FROM 'https://neo4j.com/docs/cypher-refcard/3.3/csv/artists- with-headers.csv' AS line CREATE (:Artist {name: line.Name, year: toInteger(line.Year)})</pre> Load CSV data which has headers.
<pre>USING PERIODIC COMMIT 500 LOAD CSV WITH HEADERS FROM 'https://neo4j.com/docs/cypher-refcard/3.3/csv/artists- with-headers.csv' AS line CREATE (:Artist {name: line.Name, year: toInteger(line.Year)})</pre> Commit the current transaction after every 500 rows when importing large amounts of data.
<pre>LOAD CSV FROM 'https://neo4j.com/docs/cypher-refcard/3.3/csv/artists- fieldterminator.csv' AS line FIELDTERMINATOR ';' CREATE (:Artist {name: line[1], year: toInteger(line[2])})</pre> Use a different field terminator, not the default which is a comma (with no whitespace around it).

RETURN
<pre>RETURN *</pre> Return the value of all variables.
<pre>RETURN n AS columnName</pre> Use alias for result column name.
<pre>RETURN DISTINCT n</pre> Return unique rows.
<pre>ORDER BY n.property</pre> Sort the result.
<pre>ORDER BY n.property DESC</pre> Sort the result in descending order.
<pre>SKIP \$skipNumber</pre> Skip a number of results.
<pre>LIMIT \$limitNumber</pre> Limit the number of results.
<pre>SKIP \$skipNumber LIMIT \$limitNumber</pre> Skip results at the top and limit the number of results.
<pre>RETURN count(*)</pre> The number of matching rows. See Aggregating Functions for more.

WITH
<pre>MATCH (user)-[:FRIEND]-(friend) WHERE user.name = \$name WITH user, count(friend) AS friends WHERE friends &gt; 10 RETURN user</pre> The WITH syntax is similar to RETURN. It separates query parts explicitly, allowing you to declare which variables to carry over to the next part.
<pre>MATCH (user)-[:FRIEND]-(friend) WITH user, count(friend) AS friends ORDER BY friends DESC SKIP 1 LIMIT 3 RETURN user</pre> ORDER BY, SKIP, and LIMIT can also be used with WITH.

UNION
<pre>MATCH (a)-[:KNOWS]-&gt;(b) RETURN b.name UNION MATCH (a)-[:LOVES]-&gt;(b) RETURN b.name</pre> Returns the distinct union of all query results. Result column types and names have to match.
<pre>MATCH (a)-[:KNOWS]-&gt;(b) RETURN b.name UNION ALL MATCH (a)-[:LOVES]-&gt;(b) RETURN b.name</pre> Returns the union of all query results, including duplicated rows.

MERGE
<pre>MERGE (n:Person {name: \$value}) ON CREATE SET n.created = timestamp() ON MATCH SET     n.counter = coalesce(n.counter, 0) + 1,     n.accessTime = timestamp()</pre> Match a pattern or create it if it does not exist. Use ON CREATE and ON MATCH for conditional updates.
<pre>MATCH (a:Person {name: \$value1}),       (b:Person {name: \$value2}) MERGE (a)-[r:LOVES]-&gt;(b)</pre> MERGE finds or creates a relationship between the nodes.
<pre>MATCH (a:Person {name: \$value1}) MERGE   (a)-[r:KNOWS]-&gt;(b:Person {name: \$value3})</pre> MERGE finds or creates subgraphs attached to the node.

DELETE
<pre>DELETE n, r</pre> Delete a node and a relationship.
<pre>DETACH DELETE n</pre> Delete a node and all relationships connected to it.
<pre>MATCH (n) DETACH DELETE n</pre> Delete all nodes and relationships from the database.

REMOVE
<pre>REMOVE n:Person</pre> Remove a label from n.
<pre>REMOVE n.property</pre> Remove a property.

FOREACH
<pre>FOREACH (r IN relationships(path)     SET r.marked = true)</pre> Execute a mutating operation for each relationship in a path.
<pre>FOREACH (value IN coll     CREATE (:Person {name: value}))</pre> Execute a mutating operation for each element in a list.

CALL
<pre>CALL db.labels() YIELD label</pre> This shows a standalone call to the built-in procedure db.labels to list all labels used in the database. Note that required procedure arguments are given explicitly in brackets after the procedure name.
<pre>CALL java.stored.procedureWithArgs</pre> Standalone calls may omit YIELD and also provide arguments implicitly via statement parameters, e.g. a standalone call requiring one argument input may be run by passing the parameter map {input: 'foo'}.
<pre>CALL db.labels() YIELD label RETURN count(label) AS count</pre> Calls the built-in procedure db.labels inside a larger query to count all labels used in the database. Calls inside a larger query always requires passing arguments and naming results explicitly with YIELD.

Operators	
General	DISTINCT, ., []
Mathematical	+, -, *, /, %, ^
Comparison	=, <>, <, >, <=, >=, IS NULL, IS NOT NULL
Boolean	AND, OR, XOR, NOT
String	+
List	+, IN, [x], [x .. y]
Regular Expression	=~
String matching	STARTS WITH, ENDS WITH, CONTAINS

null
<ul style="list-style-type: none"><li>null is used to represent missing/undefined values.</li><li>null is not equal to null. Not knowing two values does not imply that they are the same value. So the expression null = null yields null and not true. To check if an expression is null, use IS NULL.</li><li>Arithmetic expressions, comparisons and function calls (except coalesce) will return null if any argument is null.</li><li>An attempt to access a missing element in a list or a property that doesn't exist yields null.</li><li>In OPTIONAL MATCH clauses, nulls will be used for missing parts of the pattern.</li></ul>

Patterns
<pre>(n:Person)</pre> Node with Person label.
<pre>(n:Person:Swedish)</pre> Node with both Person and Swedish labels.
<pre>(n:Person {name: \$value})</pre> Node with the declared properties.
<pre>()-[r {name: \$value}]-( )</pre> Matches relationships with the declared properties.
<pre>(n)--&gt;(m)</pre> Relationship from n to m.
<pre>(n)--(m)</pre> Relationship in any direction between n and m.
<pre>(n:Person)-&gt;(m)</pre> Node n labeled Person with relationship to m.
<pre>(m)&lt;-[:KNOWS]-(n)</pre> Relationship of type KNOWS from n to m.
<pre>(n)-[:KNOWS :LOVES]-&gt;(m)</pre> Relationship of type KNOWS or of type LOVES from n to m.
<pre>(n)-[r]-&gt;(m)</pre> Bind the relationship to variable r.
<pre>(n)-[*1..5]-&gt;(m)</pre> Variable length path of between 1 and 5 relationships from n to m.
<pre>(n)-[*]-&gt;(m)</pre> Variable length path of any number of relationships from n to m. (See Performance section.)
<pre>(n)-[:KNOWS]-&gt;(m {property: \$value})</pre> A relationship of type KNOWS from a node n to a node m with the declared property.
<pre>shortestPath((n1:Person)-[*..6]-(n2:Person))</pre> Find a single shortest path.
<pre>allShortestPaths((n1:Person)-[*..6]-&gt;(n2:Person))</pre> Find all shortest paths.
<pre>size((n)-&gt;()-&gt;())</pre> Count the paths matching the pattern.

Lists
<pre>[ 'a', 'b', 'c' ] AS list</pre> Literal lists are declared in square brackets.
<pre>size(\$list) AS len, \$list[0] AS value</pre> Lists can be passed in as parameters.
<pre>range(\$firstNum, \$lastNum, \$step) AS list</pre> range() creates a list of numbers (step is optional), other functions returning lists are: labels(), nodes(), relationships(), filter(), extract().
<pre>MATCH p = (a)-[:KNOWS*]-&gt;( ) RETURN relationships(p) AS r</pre> The list of relationships comprising a variable length path can be returned using named paths and relationships().
<pre>RETURN matchedNode.list[0] AS value,       size(matchedNode.list) AS len</pre> Properties can be lists of strings, numbers or booleans.
<pre>list[\$idx] AS value, list[\$startIndex..\$endIdx] AS slice</pre> List elements can be accessed with idx subscripts in square brackets. Invalid indexes return null. Slices can be retrieved with intervals from start_idx to end_idx, each of which can be omitted or negative. Out of range elements are ignored.
<pre>UNWIND \$names AS name MATCH (n {name: name}) RETURN avg(n.age)</pre> With UNWIND, any list can be transformed back into individual rows. The example matches all names from a list of names.
<pre>MATCH (a) RETURN [(a)--&gt;(b) WHERE b.name = 'Bob'   b.age]</pre> Pattern comprehensions may be used to do a custom projection from a match directly into a list.
<pre>MATCH (person) RETURN person { .name, .age}</pre> Map projections may be easily constructed from nodes, relationships and other map values.



Labels
<pre>CREATE (n:Person {name: \$value})</pre> <div>Create a node with label and property.</div>
<pre>MERGE (n:Person {name: \$value})</pre> <div>Matches or creates unique node(s) with the label and property.</div>
<pre>SET n:Spouse:Parent:Employee</pre> <div>Add label(s) to a node.</div>
<pre>MATCH (n:Person)</pre> <div>Matches nodes labeled <code>Person</code>.</div>
<pre>MATCH (n:Person) WHERE n.name = \$value</pre> <div>Matches nodes labeled <code>Person</code> with the given <code>name</code>.</div>
<pre>WHERE (n:Person)</pre> <div>Checks the existence of the label on the node.</div>
<pre>labels(n)</pre> <div>Labels of the node.</div>
<pre>REMOVE n:Person</pre> <div>Remove the label from the node.</div>

Maps
<pre>{name: 'Alice', age: 38,   address: {city: 'London', residential: true}}</pre> <div>Literal maps are declared in curly braces much like property maps. Lists are supported.</div>
<pre>WITH {person: {name: 'Anne', age: 25}} AS p RETURN p.person.name</pre> <div>Access the property of a nested map.</div>
<pre>MERGE (p:Person {name: \$map.name})   ON CREATE SET p = \$map</pre> <div>Maps can be passed in as parameters and used either as a map or by accessing keys.</div>
<pre>MATCH (matchedNode:Person) RETURN matchedNode</pre> <div>Nodes and relationships are returned as maps of their data.</div>
<pre>map.name, map.age, map.children[0]</pre> <div>Map entries can be accessed by their keys. Invalid keys result in an error.</div>

Predicates
<pre>n.property &lt;&gt; \$value</pre> <div>Use comparison operators.</div>
<pre>exists(n.property)</pre> <div>Use functions.</div>
<pre>n.number &gt;= 1 AND n.number &lt;= 10</pre> <div>Use boolean operators to combine predicates.</div>
<pre>1 &lt;= n.number &lt;= 10</pre> <div>Use chained operators to combine predicates.</div>
<pre>n:Person</pre> <div>Check for node labels.</div>
<pre>variable IS NULL</pre> <div>Check if something is <code>null</code>.</div>
<pre>NOT exists(n.property) OR n.property = \$value</pre> <div>Either the property does not exist or the predicate is <code>true</code>.</div>
<pre>n.property = \$value</pre> <div>Non-existing property returns <code>null</code>, which is not equal to anything.</div>
<pre>n["property"] = \$value</pre> <div>Properties may also be accessed using a dynamically computed property name.</div>
<pre>n.property STARTS WITH 'Tob' OR n.property ENDS WITH 'n' OR n.property CONTAINS 'goodie'</pre> <div>String matching.</div>
<pre>n.property =~ 'Tob.*'</pre> <div>String regular expression matching.</div>
<pre>(n)-[:KNOWS]-&gt;(m)</pre> <div>Ensure the pattern has at least one match.</div>
<pre>NOT (n)-[:KNOWS]-&gt;(m)</pre> <div>Exclude matches to <code>(n)-[:KNOWS]-&gt;(m)</code> from the result.</div>
<pre>n.property IN [\$value1, \$value2]</pre> <div>Check if an element exists in a list.</div>

List Predicates
<pre>all(x IN coll WHERE exists(x.property))</pre> <div>Returns <code>true</code> if the predicate is <code>true</code> for all elements in the list.</div>
<pre>any(x IN coll WHERE exists(x.property))</pre> <div>Returns <code>true</code> if the predicate is <code>true</code> for at least one element in the list.</div>
<pre>none(x IN coll WHERE exists(x.property))</pre> <div>Returns <code>true</code> if the predicate is <code>false</code> for all elements in the list.</div>
<pre>single(x IN coll WHERE exists(x.property))</pre> <div>Returns <code>true</code> if the predicate is <code>true</code> for exactly one element in the list.</div>

List Expressions
<pre>size(\$list)</pre> <div>Number of elements in the list.</div>
<pre>reverse(\$list)</pre> <div>Reverse the order of the elements in the list.</div>
<pre>head(\$list), last(\$list), tail(\$list)</pre> <div><code>head()</code> returns the first, <code>last()</code> the last element of the list. <code>tail()</code> returns all but the first element. All return <code>null</code> for an empty list.</div>
<pre>[x IN list WHERE x.prop &lt;&gt; \$value   x.prop]</pre> <div>Combination of filter and extract in a concise notation.</div>
<pre>extract(x IN list   x.prop)</pre> <div>A list of the value of the expression for each element in the original list.</div>
<pre>filter(x IN list WHERE x.prop &lt;&gt; \$value)</pre> <div>A filtered list of the elements where the predicate is <code>true</code>.</div>
<pre>reduce(s = "", x IN list   s + x.prop)</pre> <div>Evaluate expression for each element in the list, accumulate the results.</div>

CASE
<pre>CASE n.eyes   WHEN 'blue' THEN 1   WHEN 'brown' THEN 2   ELSE 3 END</pre> <div>Return <code>THEN</code> value from the matching <code>WHEN</code> value. The <code>ELSE</code> value is optional, and substituted for <code>null</code> if missing.</div>
<pre>CASE   WHEN n.eyes = 'blue' THEN 1   WHEN n.age &lt; 40 THEN 2   ELSE 3 END</pre> <div>Return <code>THEN</code> value from the first <code>WHEN</code> predicate evaluating to <code>true</code>. Predicates are evaluated in order.</div>

Functions
<pre>coalesce(n.property, \$defaultValue)</pre> <div>The first non-<code>null</code> expression.</div>
<pre>timestamp()</pre> <div>Milliseconds since midnight, January 1, 1970 UTC.</div>
<pre>id(nodeOrRelationship)</pre> <div>The internal id of the relationship or node.</div>
<pre>toInteger(\$expr)</pre> <div>Converts the given input into an integer if possible; otherwise it returns <code>null</code>.</div>
<pre>toFloat(\$expr)</pre> <div>Converts the given input into a floating point number if possible; otherwise it returns <code>null</code>.</div>
<pre>toBoolean(\$expr)</pre> <div>Converts the given input into a boolean if possible; otherwise it returns <code>null</code>.</div>
<pre>keys(\$expr)</pre> <div>Returns a list of string representations for the property names of a node, relationship, or map.</div>
<pre>properties({\$expr})</pre> <div>Returns a map containing all the properties of a node or relationship.</div>

Path Functions
<pre>length(path)</pre> <div>The number of relationships in the path.</div>
<pre>nodes(path)</pre> <div>The nodes in the path as a list.</div>
<pre>relationships(path)</pre> <div>The relationships in the path as a list.</div>
<pre>extract(x IN nodes(path)   x.prop)</pre> <div>Extract properties from the nodes in a path.</div>

Spatial Functions
<pre>point({x: {x}, y: {y}})</pre> <div>Returns a point in a 2D coordinate system.</div>
<pre>distance(point({x: {x1}, y: {y1}}), point({x: {x2}, y: {y2}}))</pre> <div>Returns a floating point number representing the geodesic distance between two points.</div>

Mathematical Functions
<pre>abs(\$expr)</pre> <div>The absolute value.</div>
<pre>rand()</pre> <div>Returns a random number in the range from 0 (inclusive) to 1 (exclusive), <code>[0,1)</code>. Returns a new value for each call. Also useful for selecting a subset or random ordering.</div>
<pre>round(\$expr)</pre> <div>Round to the nearest integer; <code>ceil()</code> and <code>floor()</code> find the next integer up or down.</div>
<pre>sqrt(\$expr)</pre> <div>The square root.</div>
<pre>sign(\$expr)</pre> <div>0 if zero, -1 if negative, 1 if positive.</div>
<pre>sin(\$expr)</pre> <div>Trigonometric functions also include <code>cos()</code>, <code>tan()</code>, <code>cot()</code>, <code>asin()</code>, <code>acos()</code>, <code>atan()</code>, <code>atan2()</code>, and <code>haversin()</code>. All arguments for the trigonometric functions should be in radians, if not otherwise specified.</div>
<pre>degrees(\$expr), radians(\$expr), pi()</pre> <div>Converts radians into degrees; use <code>radians()</code> for the reverse, and <code>pi()</code> for <math>\pi</math>.</div>
<pre>log10(\$expr), log(\$expr), exp(\$expr), e()</pre> <div>Logarithm base 10, natural logarithm, e to the power of the parameter, and the value of e.</div>

String Functions
<pre>toString(\$expression)</pre> <div>String representation of the expression.</div>
<pre>replace(\$original, \$search, \$replacement)</pre> <div>Replace all occurrences of <code>search</code> with <code>replacement</code>. All arguments must be expressions.</div>
<pre>substring(\$original, \$begin, \$sublength)</pre> <div>Get part of a string. The <code>sublength</code> argument is optional.</div>
<pre>left(\$original, \$sublength), right(\$original, \$sublength)</pre> <div>The first part of a string. The last part of the string.</div>
<pre>trim(\$original), lTrim(\$original), rTrim(\$original)</pre> <div>Trim all whitespace, or on the left or right side.</div>
<pre>toUpperCase(\$original), toLowerCase(\$original)</pre> <div>UPPERCASE and lowercase.</div>
<pre>split(\$original, \$delimiter)</pre> <div>Split a string into a list of strings.</div>
<pre>reverse(\$original)</pre> <div>Reverse a string.</div>
<pre>size(\$string)</pre> <div>Calculate the number of characters in the string.</div>

Relationship Functions
<pre>type(a_relationship)</pre> <div>String representation of the relationship type.</div>
<pre>startNode(a_relationship)</pre> <div>Start node of the relationship.</div>
<pre>endNode(a_relationship)</pre> <div>End node of the relationship.</div>
<pre>id(a_relationship)</pre> <div>The internal id of the relationship.</div>

Aggregating Functions
<pre>count(*)</pre> <div>The number of matching rows.</div>
<pre>count(variable)</pre> <div>The number of non-<code>null</code> values.</div>
<pre>count(DISTINCT variable)</pre> <div>All aggregating functions also take the <code>DISTINCT</code> operator, which removes duplicates from the values.</div>
<pre>collect(n.property)</pre> <div>List from the values, ignores <code>null</code>.</div>
<pre>sum(n.property)</pre> <div>Sum numerical values. Similar functions are <code>avg()</code>, <code>min()</code>, <code>max()</code>.</div>
<pre>percentileDisc(n.property, \$percentile)</pre> <div>Discrete percentile. Continuous percentile is <code>percentileCont()</code>. The percentile argument is from 0.0 to 1.0.</div>
<pre>stDev(n.property)</pre> <div>Standard deviation for a sample of a population. For an entire population use <code>stDevP()</code>.</div>

INDEX
<pre>CREATE INDEX ON :Person(name)</pre> <div>Create an index on the label <code>Person</code> and property <code>name</code>.</div>
<pre>MATCH (n:Person) WHERE n.name = \$value</pre> <div>An index can be automatically used for the equality comparison. Note that for example <code>toLower(n.name) = \$value</code> will not use an index.</div>
<pre>MATCH (n:Person) WHERE n.name IN [\$value]</pre> <div>An index can automatically be used for the <code>IN</code> list checks.</div>
<pre>MATCH (n:Person) USING INDEX n:Person(name) WHERE n.name = \$value</pre> <div>Index usage can be enforced when Cypher uses a suboptimal index, or more than one index should be used.</div>
<pre>DROP INDEX ON :Person(name)</pre> <div>Drop the index on the label <code>Person</code> and property <code>name</code>.</div>

CONSTRAINT
<pre>CREATE CONSTRAINT ON (p:Person)   ASSERT p.name IS UNIQUE</pre> <div>Create a unique property constraint on the label <code>Person</code> and property <code>name</code>. If any other node with that label is updated or created with a <code>name</code> that already exists, the write operation will fail. This constraint will create an accompanying index.</div>
<pre>DROP CONSTRAINT ON (p:Person)   ASSERT p.name IS UNIQUE</pre> <div>Drop the unique constraint and index on the label <code>Person</code> and property <code>name</code>.</div>
<pre>CREATE CONSTRAINT ON (p:Person)   ASSERT exists(p.name)</pre> <div>Create a node property existence constraint on the label <code>Person</code> and property <code>name</code>. If a node with that label is created without a <code>name</code>, or if the <code>name</code> property is removed from an existing node with the <code>Person</code> label, the write operation will fail.</div>
<pre>DROP CONSTRAINT ON (p:Person)   ASSERT exists(p.name)</pre> <div>Drop the node property existence constraint on the label <code>Person</code> and property <code>name</code>.</div>
<pre>CREATE CONSTRAINT ON ()-[l:LIKED]-()   ASSERT exists(l.when)</pre> <div>Create a relationship property existence constraint on the type <code>LIKED</code> and property <code>when</code>. If a relationship with that type is created without a <code>when</code>, or if the <code>when</code> property is removed from an existing relationship with the <code>LIKED</code> type, the write operation will fail.</div>
<pre>DROP CONSTRAINT ON ()-[l:LIKED]-()   ASSERT exists(l.when)</pre> <div>Drop the relationship property existence constraint on the type <code>LIKED</code> and property <code>when</code>.</div>

Performance
<ul style="list-style-type: none"><li>Use parameters instead of literals when possible. This allows Cypher to re-use your queries instead of having to parse and build new execution plans.</li><li>Always set an upper limit for your variable length patterns. It's easy to have a query go wild and touch all nodes in a graph by mistake.</li><li>Return only the data you need. Avoid returning whole nodes and relationships — instead, pick the data you need and return only that.</li><li>Use <code>PROFILE</code> / <code>EXPLAIN</code> to analyze the performance of your queries. See <a href="#">Query Tuning</a> for more information on these and other topics, such as planner hints.</li></ul>