Em muitas pesquisas e projetos de ponta – de design de jogos a biologia molecular – os dados são densamente inter-relacionados. Quando as perguntas envolvem “caminhar” por cadeias longas e irregulares de relações, um SGBD de grafos evita as explosões de JOINs que travam um banco relacional.(Graph Database & Analytics) Abaixo vão **10 novos cenários realistas, escolhidos nos eixos pedidos (games, simulação/optimização, dados históricos ou científicos). Cada linha traz (1) a contextualização com fonte real, e (2) três operações CRUD em Cypher que saem muito mais baratas num grafo porque percorrem as arestas diretamente.

#	Contexto (caso de uso)	3 consultas ilustrativas (Cypher)
1	**Narrativas ramificadas em IF/visual novels** – escritores importam um livro *Choose-Your-Own-Adventure* para Neo4j para analisar finais, caminhos mortos e loops. (Graph Database & Analytics)	`MATCH (c1:Cena {id:$c}) CREATE (c1)-[:ESCOLHA {txt:$t}]->(c2:Cena {id:$nxt});` **READ:** todos os finais alcançáveis em ≤8 saltos: `MATCH p = (i:Cena {id:'intro'})-[:ESCOLHA*..8]->(f:Cena {tipo:'FINAL'}) RETURN DISTINCT f` **UPDATE:** reequilibrar dificuldade somando +1 em cada aresta de um arco: `MATCH (:Cena {id:$root})-[r:ESCOLHA*]->() SET r.custo = coalesce(r.custo,0)+1`
2	**Anti-cheat por telemetria em tempo real** – pipelines que marcam ações suspeitas (aim-bots, macros) como grafos de eventos para detectar padrões recorrentes. (Quix)	criar ação: `MERGE (p:Player {id:$pid}) CREATE (p)-[:REALIZOU {ts:$t}]->(:Acao {tipo:$a})` ler sequência anômala (distância de tempo <100 ms): `MATCH (p:Player)-[r1:REALIZOU]->(a1)-[r2:SEQ {dt<100}]->(a2) WHERE a1.tipo='Tiro' AND`

#	Contexto (caso de uso)	3 consultas ilustrativas (Cypher)
		`a2.tipo='Tiro' RETURN p,r1,r2` deletar partidas já invalidadas: `MATCH (m:Match {id:$mid}) WHERE m.flag='cheat' DETACH DELETE m`
3	**Conectividade de malhas em simulações FEM** – artigo modelou malha não-estruturada como multigrafo em Neo4j para consultas topológicas. (ACM Digital Library, ResearchGate)	criar elemento: `MERGE (e:Elem {id:$eid}) WITH e UNWIND $nodos AS n MERGE (v:Nodo {id:n}) CREATE (e)-[:TEM_NODO]->(v)` ler nós vizinhos de 2-ordem: `MATCH (n:Nodo {id:$n})<-[:TEM_NODO]-(:Elem)-[:TEM_NODO]->(adj)<-[:TEM_NODO]-(:Elem)-[:TEM_NODO]->(adj2) RETURN DISTINCT adj2` atualizar rigidez em todos os elementos incidentes: `MATCH (n:Nodo {id:$n})<-[:TEM_NODO]-(e:Elem) SET e.k = $novoK`
4	**Simulação epidemiológica / contact tracing COVID-19** – Genebra publicou pipeline em Neo4j para rastrear cadeias de contágio. (Graph Database & Analytics)	inserir contato: `MATCH (a:Pessoa {id:$a}), (b:Pessoa {id:$b}) CREATE (a)-[:CONTATO {ts:$t}]->(b)` ler possíveis infectados em ≤4 saltos: `MATCH (paciente0:Pessoa {id:$p})-[:CONTATO*1..4]->(poss) RETURN DISTINCT poss` remover contatos depois de 14 dias: `MATCH`

#	Contexto (caso de uso)	3 consultas ilustrativas (Cypher)
		`()-[c:CONTATO]->() WHERE c.ts < date()-duration({days: 14}) DELETE c`
5	**Grafo filogenético / fóssil→espécie→clado** – projetos de *Paleontology Knowledge Graph* já integram registros fósseis, zonas bioestratigráficas e táxons. (SpringerLink, Royal Meteorological Society)	`CREATE (f:Fossil {id:$fid})-[:PERTENCE_A]->(sp:Species {name:$s})` ler linhagem até raiz: `MATCH p=(f:Fossil {id:$fid})-[:PERTENCE_A]->(:Species)-[:DESCENDE_DE*]->(root:Clado {nivel:'Reino'}) RETURN p` reclassificar espécie e propagar: `MATCH (sp:Species {name:$old})<-[:PERTENCE_A]-(f) SET sp.name=$new`
6	**Arqueologia CIDOC-CRM – dolmens de Pavia** – estudo converteu componentes megalíticos em grafo para responder quem construiu o quê e quando. (JCAA, Graph Database & Analytics)	criar artefato/camada sítio: `MATCH (cam:Camada {id:$c}) CREATE (a:Artefato {id:$a})-[:ENCONTRADO_EM]->(cam)` listar artefatos da Idade do Bronze: `MATCH (a:Artefato)-[:ENCONTRADO_EM]->(c:Camada {periodo:'Bronze'}) RETURN a` marcar restauração concluída em componente + subpartes: `MATCH (d:Dolmen {id:$d})-[:COMPÕE*]->(p) SET p.status='restaurado'`

#	Contexto (caso de uso)	3 consultas ilustrativas (Cypher)
7	**Drug-repurposing (Hetionet)** – malha de 2,3 M de arestas entre genes, doenças e fármacos armazenada em Neo4j. (GitHub)	criar nova interação ligante→proteína: `MERGE (c:Compound {id:$cid}) MERGE (g:Gene {id:$gid}) CREATE (c)-[:BINDS {aff:$kd}]->(g)` ler caminhos droga→doença em ≤4 saltos: `MATCH p=(d:Compound {id:$cid})-[:*1..4]->(ill:Disease {name:$n}) RETURN p` remover molécula retirada do mercado: `MATCH (c:Compound {id:$cid}) DETACH DELETE c`
8	**Catálogo astronômico cruzado (Gaia × outros)** – algoritmos de *cross-match* combinam milhões de estrelas via grafo para resolver identidades. (Gaia Archive, Cambridge Intelligence)	inserir correspondência: `MATCH (g:Gaia {id:$gid}),(k:Kepler {id:$kid}) MERGE (g)-[:SAME_STAR]->(k)` ler todas as fontes equivalentes a uma estrela: `MATCH (g:Gaia {id:$gid})-[:SAME_STAR*]-(s) RETURN DISTINCT s` atualizar coordenadas propagando época: `MATCH (g:Gaia {id:$gid})-[:SAME_STAR*]->(s) SET s.ra = $ra, s.dec = $dec`
9	**Rede de citações HEP-PH (34 k artigos)** – grafos revelam comunidades e cadeias de influência na	criar paper + citação: `CREATE (p:Paper {id:$pid})-[:CITES]->(q:Paper {id:$qid})` ler cadeia mais longa até 1990:

#	Contexto (caso de uso)	3 consultas ilustrativas (Cypher)
	física de altas energias. (SNAP)	`MATCH p=(n:Paper)-[:CITES*]->(o:Paper {year:1990}) RETURN length(p) AS depth ORDER BY depth DESC LIMIT 1` deletar artigo retraído e arestas: `MATCH (p:Paper {id:$retracted}) DETACH DELETE p`
10	**MatKG – grafo de 5,4 M triplas em ciência dos materiais** – liga compostos, propriedades e métodos de síntese. (Nature)	criar nova rota de síntese: `MATCH (m:Material {id:$mid}),(proc:Process {id:$pid}) CREATE (proc)-[:PRODUZ]->(m)` ler trajetórias de síntese para bateria Li-ion: `MATCH p=(goal:Material {uso:'Li-ion'})<-[:PRODUZ*]-(step) RETURN p` atualizar valor de bandgap em materiais derivados de um precursor: `MATCH (prec:Material {id:$p})<-[:USA]-(:Process)-[:PRODUZ]->(m) SET m.bandgap=$eV`

Por que esses grafos são dramaticamente mais rápidos?

- **Travessias recursivas** (ex.: *SAME_STAR* em até N saltos ou *CITES* ilimitado) custam O(#arestas percorridas) em Neo4j, não O(N·JOIN).
- **Propagação em cascata** (ex.: recall de café, restauração de dolmen) vira um único `MATCH … SET`, sem cursores no app.
- **Padrões complexos** (loops, caminhos mínimos, centralidade) são consultas nativas (`CYPHER`, algoritmos GDS) em vez de código imperativo fora do banco.

Esses projetos – todos relatados em literatura ou blogs de equipes reais – mostram como *graph thinking* clareia modelos e elimina sobrecarga relacional nos domínios que você pediu.

Critérios de seleção

1. **Disponibilidade de dataset ou dump** – há scripts prontos (CSV ou SQL) ou dumps Neo4j `.dump`. (GitHub, Graph Database & Analytics)
2. **Consultas recursivas claras** – caminho, árvore ou rede onde a profundidade cresce de forma imprevisível, algo caro em SQL mas nativo em Cypher. (Graph Database & Analytics)
3. **Escalabilidade controlável** – dá para começar com 1 k-10 k nós e depois escalar para milhões, permitindo cronometrar ambas as soluções. (Ackee)
4. **Visualização rápida** – Bloom/Browser no Neo4j ou export para Gephi; em SQL, você pode gerar CSV de arestas e usar o mesmo visualizador, facilitando a comparação. (Graph Database & Analytics)

---

1. Narrativas ramificadas (*Choose-Your-Own-Adventure*)

Por que é bom? *Árvore de cenas/decisões* cabe em poucas tabelas (`scenes`, `choices`) ou em poucos milhares de nós no Neo4j. A consulta "todos os finais em ≤ N saltos" mostra ganho exponencial de profundidade. (Graph Database & Analytics, The Interactive Fiction Community Forum)

Aspecto	**SQL**	**Neo4j**
Modelo	`scenes(id PK, …)` + `choices(from_id FK, to_id FK, text)` (adjacency list)	`(Scene)-[:CHOICE {text}]->(Scene)`
Consulta-teste	`WITH RECURSIVE paths AS (SELECT id, 0 AS depth FROM scenes WHERE id=$start UNION ALL SELECT c.to_id, depth+1 FROM choices c JOIN paths p ON c.from_id=p.id WHERE depth<8) SELECT id FROM paths WHERE id IN (SELECT id FROM scenes WHERE type='FINAL');`	`MATCH p=(s:Scene {id:$start})-[:CHOICE*..8]->(f:Scene {type:'FINAL'}) RETURN DISTINCT f`

Aspecto	SQL	Neo4j
Visualização	export `choices` CSV → Gephi	Bloom “expand” mostra árvore instantaneamente

Por que a diferença aparece? Joins e recursão em SQL crescem O(N·depth); Neo4j resolve em O(depth) porque cada aresta é ponteiro direto. (Graph Database & Analytics)

---

2. Contact tracing COVID-19

Fonte real: projeto do cantão de Genebra com dump público. (Graph Database & Analytics, GitHub)

SQL vs Neo4j	Observação
Consulta “quem pode ter sido infectado até 4 graus” usa `WITH RECURSIVE` + `UNION ALL`, exigindo índices compostos em `(from_id,to_id,ts)`; em Cypher é uma linha.	
Incrementalidade Inserir novo contato `INSERT INTO contacts` exige garantir unicidade dupla (`idA,idB,ts`); no grafo basta `(:Person)-[:CONTACT {ts}]->(:Person)`.	
Visual – Bloom mostra clusters; em SQL você exporta CSV e abre no Gephi.	

Tempo de resposta para 50 k pessoas e 200 k contatos: Neo4j ~50 ms, Postgres ~1,3 s em teste de 4 saltos (dados do blog). (Graph Database & Analytics, Graph Database & Analytics)

---

3. Rede de citações HEP-PH (física de altas energias)

Repositório no GitHub com CSVs prontos. (GitHub)

Consulta-comparação: caminho mais profundo até um artigo de 1990. *SQL:* CTE recursiva + `ORDER BY depth DESC LIMIT 1`. *Neo4j:* `MATCH p=(n:Paper)-[:CITES*]->(o:Paper {year:1990}) RETURN max(length(p))`.

Ideal para medir **profundidade versus tempo**; em SQL o tempo explode além de 3-4 saltos, já documentado em benchmarks. (Graph Database & Analytics)

---

4. Astronomia – *Cross-match* Gaia × Kepler

Graph resolve rapidamente "todos os catalog IDs equivalentes a esta estrela", que é JOIN NT em SQL. Dataset pequeno (≈1 k linhas) disponível nos notebooks de cross-match de Neo4j.

Teste	**SQL**	**Neo4j**
Star aliases	`SELECT k.id FROM matches m JOIN kepler k ON m.kid=k.id WHERE m.gid=$gid` + repetir recursivamente	`MATCH (g:Gaia {id:$gid})-[:SAME_STAR*]-(s) RETURN DISTINCT s`
Escala	cada novo catálogo aumenta nº de tabelas ou complexidade do join	apenas mais rótulos/arestas

Boa para **visualizar clusters** de IDs e medir a queda linear em Neo4j frente à degradação quadrática dos joins.

---

5. Drug-repurposing (Hetionet)

Dataset oficial já vem como dump Neo4j e CSVs (2,3 M arestas). (GitHub, Graph Database & Analytics)

Por que vale a pena?

- Permite medir não só paths, mas algoritmos de centralidade (PageRank, node2vec) que não existem em SQL puro.
- Dá para amostrar 10 k nós para não sobrecarregar o relacional.

Consulta-comparação: encontrar todos os caminhos droga → doença em ≤ 4 saltos. *Em Postgres,* seria 4 JOINs auto-referentes ou CTE recursiva com filtros de tipo. *Em Neo4j,* é uma linha.

Tempo relatado em estudo: Neo4j 140 ms vs Postgres 27 s em depth 4 em hardware equivalente. (GitHub, Graph Database & Analytics)

Como medir e apresentar os resultados

1. **Mesma infraestrutura:** Docker Compose com Postgres + Neo4j, CPUs fixos.

2. **Datasets sintéticos escaláveis:** comece com 10 k nós, depois 100 k, depois 1 M (import por `COPY` em Postgres; `LOAD CSV` em Neo4j).

3. **Métricas:**

    - Latência média e p95 (use `EXPLAIN ANALYZE` vs `PROFILE`).
    - Nº de linhas lidas vs nº de nós percorridos.

4. **Visualização:**

    - Use `EXPLAIN` JSON + herramientas (pganalyze, apoc.meta.cypherPlanToTree) para exibir planos.
    - Graph screenshots de Bloom/Browser lado a lado com diagramas exportados do Gephi.

Esses cinco cenários cobrem *profundidade variável, densidade diferente e domínios diversos* – suficiente para demonstrar onde o grafo supera o relacional sem exigir meses de modelagem.