The burden of workload-driven schema design in NoSQL databases

Enrico Gallinucci
enrico.gallinucci@unibo.it

Paris, June 2026

Who am I

Enrico Gallinucci, Ph.D.

- Tenure Track Researcher @ DISI, UniBO
- Website - LinkedIn

Main teaching

- **Big Data** since 2017
- **Business Intelligence** (Module 2) since 2021
- BI & Data pipelines @ Bologna Business School

My research group

The Business Intelligence Group

- Active since 1997
- Studies methodologies, techniques, and technologies with the broad scope of **data analysis**
- Key drivers: promoting a data-driven culture and digital transformation, supporting simpler/faster/effective decision-making

Traditional research

- Business Intelligence & Data Warehousing
- Social BI -- Linked Open Data -- Trajectory Mining

Current research topics

- Big Data & NoSQL optimization -- Knowledge Graphs & Time series
- LLMs and Hybrid RAG for data analysis -- Precision Agriculture

Group composition

- 2 professors (Rizzi S. and Golfarelli M.), 2 senior researchers (Francia M. and me), 1 Ph.D. student (Pasini M.), 1 research assistant (Baiardi A.)

My research group

European funding

- PANDA (pattern management in DM)
- ENPADASI (EU Nutritional Phenotype Assessment and Data Sharing Initiative)
- TOREADOR (As-a-service Big Data Analytics)
- WeLaser (Laser-based Robotic Weeding)
- PNRR (Collaborative Data Platform for Precision Farming)

Public funding

- D2I (integration and mining of heterogeneous DBs)
- WISDOM (ontology-enhanced web searching)
- WebPolEU (Comparing Social Media and Political Participation across EU)
- GenData2020 (data-centric genomic computing)
- DyNamiTE (Digital fightiNg Tax Evasion)
- MO.RE.Farming (Big Data for Precision Farming)
- INNOFRUVE (Industrial research and innovation in the fruit sector)
- AgroBigDataScience (Big Data for Precision Farming)
- AI4AgriHub (Artificial Intelligence for Precision Farming)

Private funding

- Data Mining in the Fashion Field with Valentino
- Set-up of a Social Business Intelligence framework with Amadori s.p.a.
- Feasibility study for a Social Business Intelligence system with DOXA
- Anomaly detection in the gas network with HERA spa
- Harnessing Wellness Knowledge with Technogym
- Methodological and Scientific Support to several Public bodies with Ministry of Justice, Economy and Finance
- Vaccine monitoring with Regione Veneto & ONIT
- Intelligent Monitoring Systems for Critical Environments with Leonardo-Finmeccanica
- Data-driven budgeting with Teddy
- Digital Transformation with BRT, PLT Energia, SCM
- Digital Auction Optimization with EasyMarket
- Prescriptive Watering with Agrintesa, Unapera
- Growth Value Model with Technogym
- Hybrid RAG on Knowledge Graphs with Technogym

My (NoSQL) research topics

Analyzing data under different levels of heterogeneity

- Collection level
- Multistore level
- Stream level

Studying workload-driven schema designs

- Analyzing the impact of workload (and domain) changes
- Measuring the cost of a schema design
- Exploring schema design alternatives and recommending one

Aggregate-oriented data model

Document-based databases encourage data modeling to be **aggregate-oriented**

- Joins are not supported or discouraged
- Collections contain *aggregates*

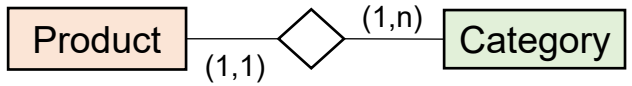
```
{ id: 1,
  name: “Water”,
  price: 1.5,
  category: “Beverage” }
```

```
{ id: 1,
  name: “Cola”,
  price: 3.0,
  category: “Beverage” }
```

Why/When

- Joins are expensive
- Latency is the first-citizen
- Storage is cheap
- Redundancy is not an issue

```
{ id: “C1”,
  name: “Beverage”,
  products: [
    { id: 1,
      name: “Water”,
      price: 1.5 },
    { id: 2,
      name: “Cola”,
      price: 3.0 }
 ] }
```



Aggregate-oriented data model

Document-based databases encourage data modeling to be **aggregate-oriented**

- Joins are not supported or discouraged
- Collections contain *aggregates*

Query: *get Price and Category of a Product*

```
{ id: 1,
  name: “Water”,
  price: 1.5,
  category: “Beverage” }
```

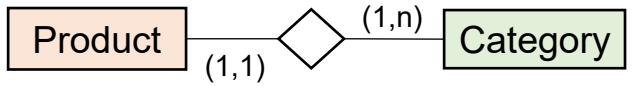
```
{ id: 1,
  name: “Cola”,
  price: 3.0,
  category: “Beverage” }
```

Problem: database design is **workload-driven**

- The structure of the aggregates heavily depends on the queries

```
{ id: “C1”,
  name: “Beverage”,
  products: [
    { id: 1,
      name: “Water”,
      price: 1.5 },
    { id: 2,
      name: “Cola”,
      price: 3.0 }
 ] }
```

Query: *get Name and Products of a Category*



Issues with workload-driven DB design

1. **What if the domain and/or the workload change?**
2. What is the best design for a workload?
   a. Which criteria define a design as the *best* one?
   b. How do we obtain it?

Analyzing the impact of workload (and domain) changes

Enrico Gallinucci, Matteo Golfarelli, Wafaa Radwan, Gabriel Zarate, Alberto Abelló
Impact Study of NoSQL Refactoring in SkyServer Database
DOLAP 2025: 1-11 **(Best Paper)**
https://ceur-ws.org/Vol-3931/paper1.pdf

Enrico Gallinucci, Matteo Golfarelli, Wafaa Radwan, Gabriel Zarate, Alberto Abelló
Impact study of incremental NoSQL refactoring in SkyServer database
Inf. Syst. 140: 102743 (2026)
https://doi.org/10.1016/j.is.2026.102743

Alberto Abelló, Enrico Gallinucci
DORM: Dynamic Object-Relational Mapping
DOLAP 2026: 1-17
https://ceur-ws.org/Vol-4186/paper1.pdf

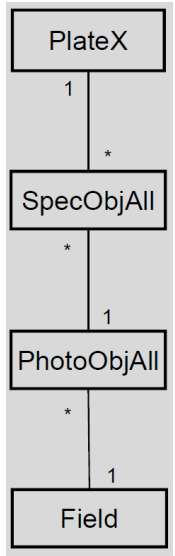
The SkyServer case study

Publicly available relational database designed to map the cosmos

- Web interface to query and download data
- SkyServer Traffic Log: captures statistics on SQL queries executed since 2003

Main terminology

- PlateX: aluminum plate used to in telescope settings
- SpecObj: measurement captured for each PhotoObj with a plate
- **PhotoObj: astronomical object observed in a Field**
- Field: region of the sky

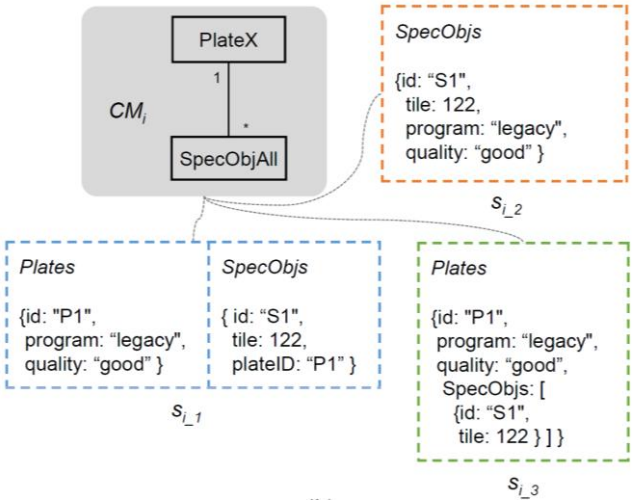


SkyServer catalog: https://skyserver.sdss.org/dr18/MoreTools/browser

Motivation - Domain evolves

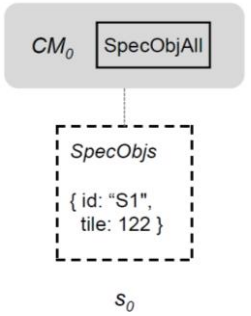
Literature on schema evolution says:

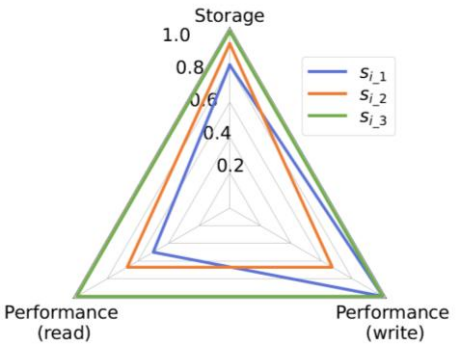
1. Schema changes depend on domain and application
2. No full knowledge of schema information at design time
3. Conditions to modify schemas can mature at any time
4. Reluctance to change a mature database schema



Consequences:

- Complex schemas built from uncertainties
- Missed opportunities
- Antipatterns





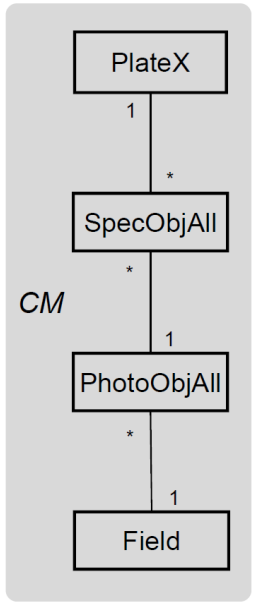
Motivation - Workload evolves

Literature on workload evolution says:

1. Query frequencies/ratios easily change
   - Cyclic patterns
   - Occasional spikes
   - Stable changes
2. Queries change
   - Reformulation due to domain evolution
   - Addition/Removal of queries

Consequences:

- Conditions that motivated the initial schema design disappear
- Initial optimizations can backfire



s_1

Plates

```
{id: "P1",
 program: "legacy",
 quality: "good" }
```

Fields

```
{ id: "F1",
 nTotal: 923,
 photoObjs: [ {
  id: "O1",
  type: "Star"
  specObjs: [ {
   id: "S1",
   waveMin: 3806,
   waveMax: 9202,
   plate_id: "P1",
   plate_quality: "good" } ] } ] }
```

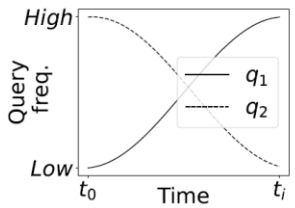
s_2

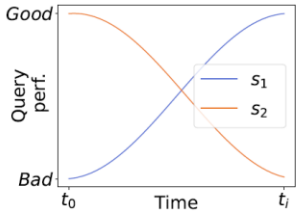
Plates

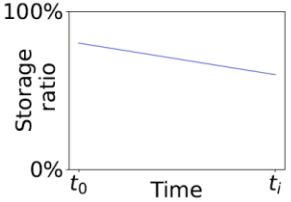
```
{id: "P1",
 program: "legacy",
 quality: "good",
 specObjs: [ {
    id: "S1",
    waveMin: 3806,
    waveMax: 9202,
    photoObj_id: "O1",
    photoObj_type: "Star"} ] }
```

Fields

```
{ id: "F1",
 nTotal: 923,
 photoObjs: [ {
  id: "O1",
  type: "Star"} ] }
```



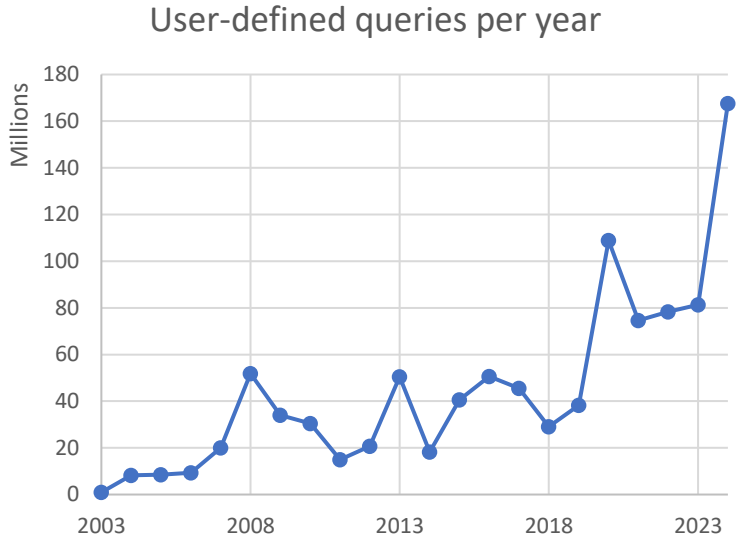


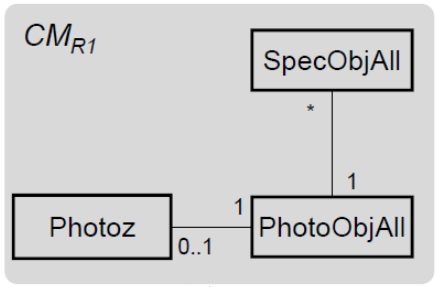


Evolution in SkyServer

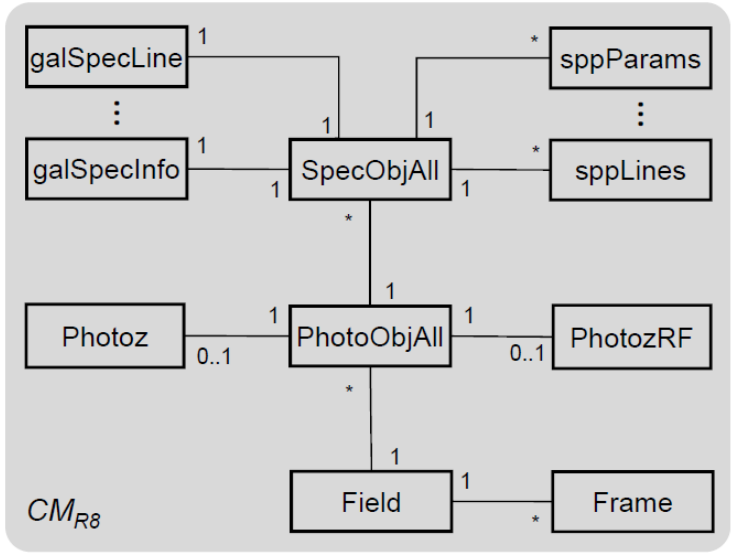
Domain evolution: 18 releases over 22 years

Workload evolution: 1 billion queries since 2003

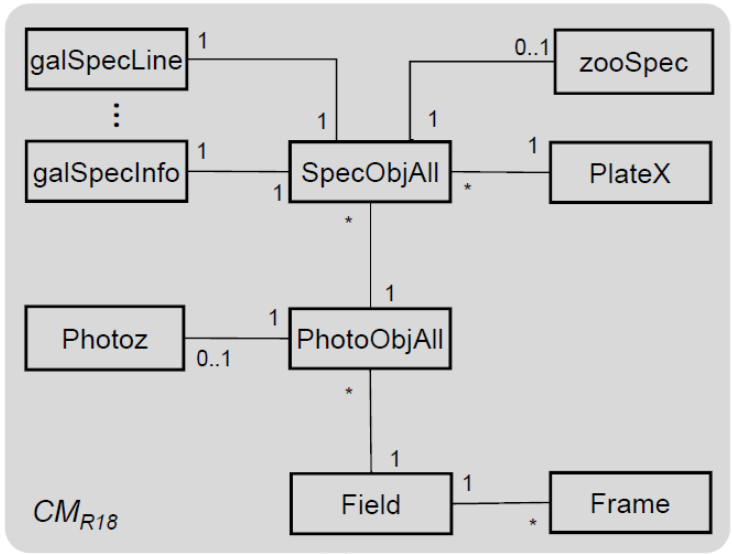




(a) R1



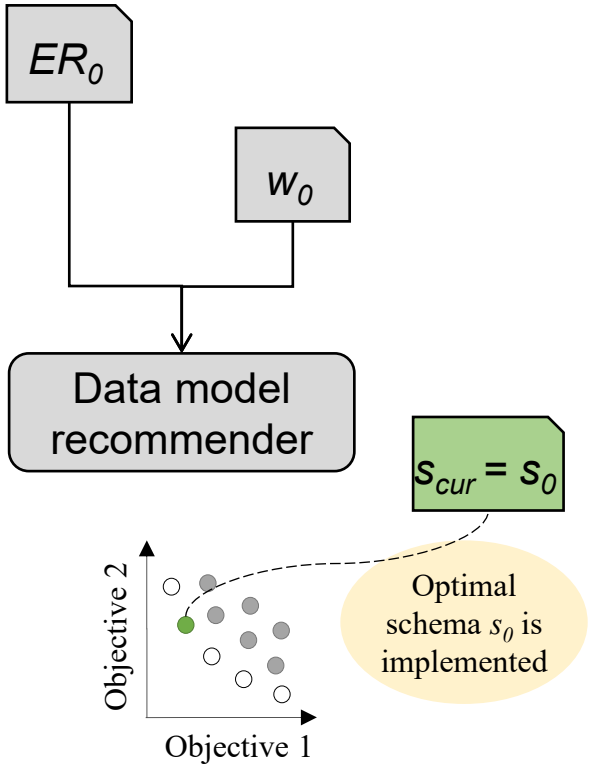
(b) R8



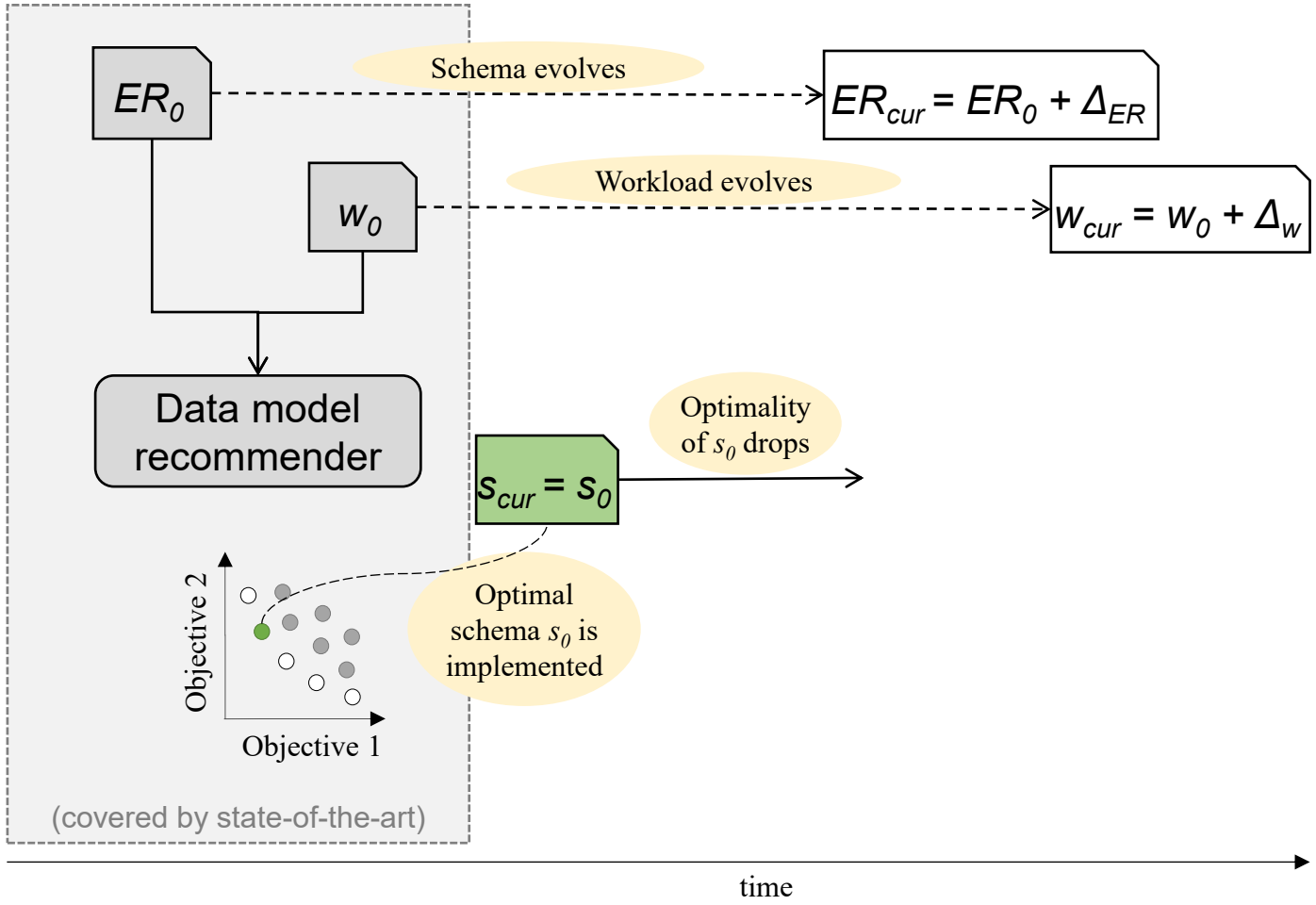
(c) R18

2003 — 2013 — 2023

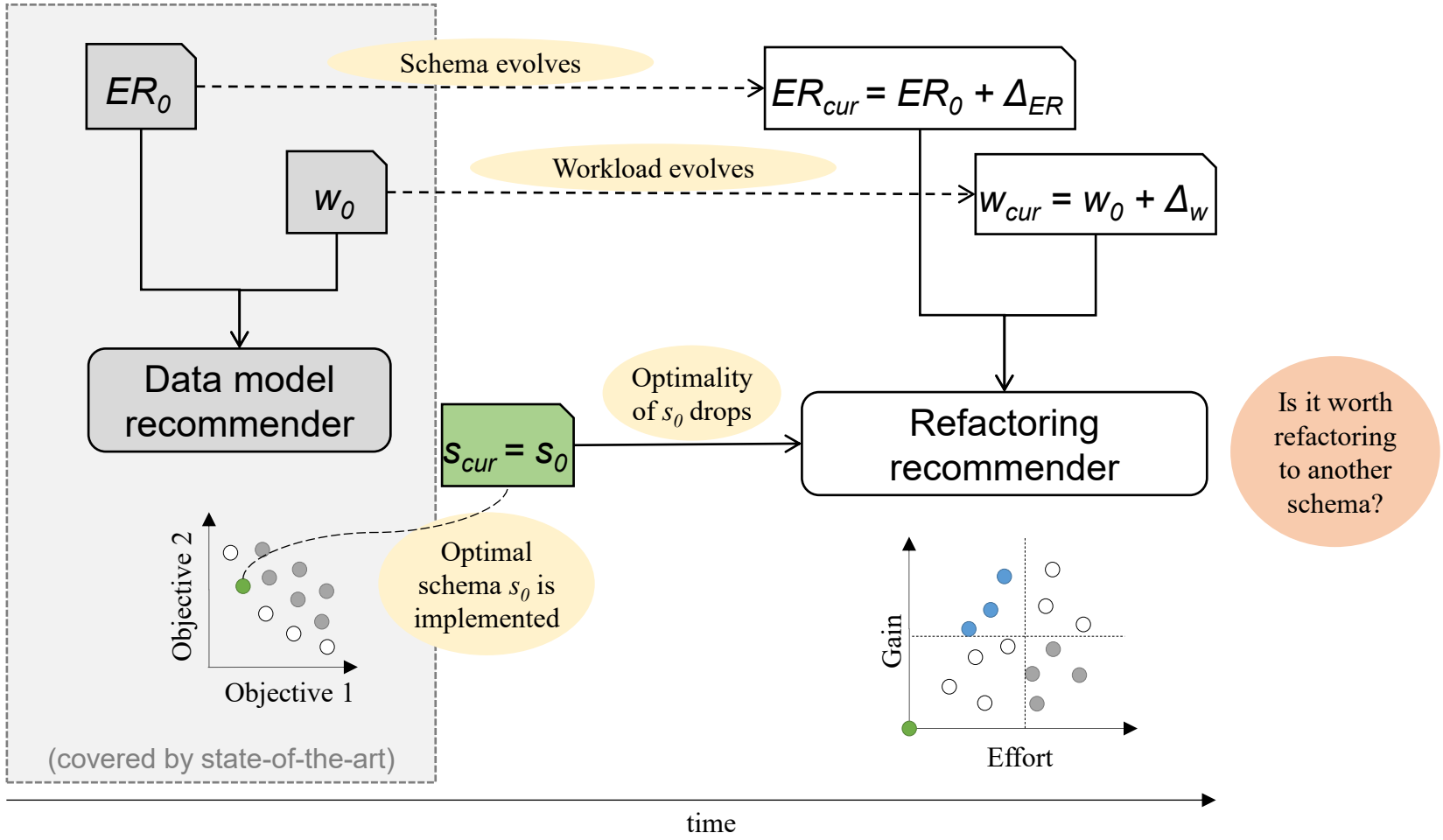
Enter (NoSQL) database refactoring



Enter (NoSQL) database refactoring

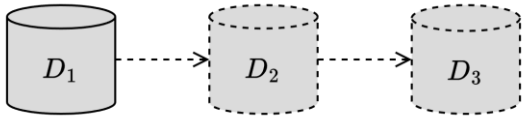


Enter (NoSQL) database refactoring



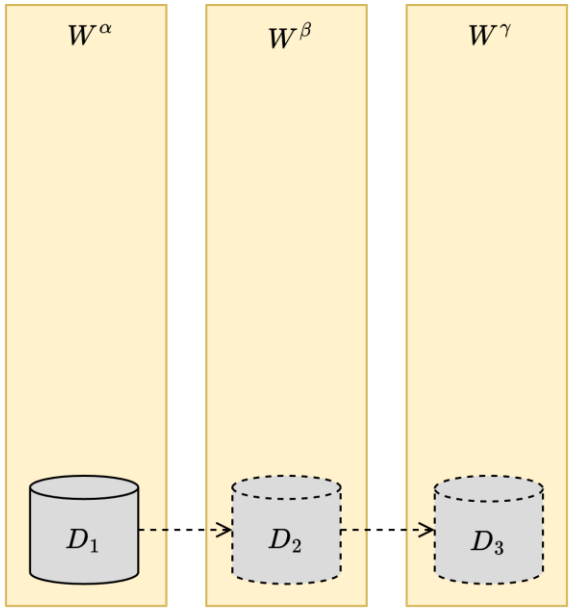
Experimental setting (I)

- Identify and download snapshots
- Upload to PostgreSQL as flat JSON (D_1, D_2, D_3)



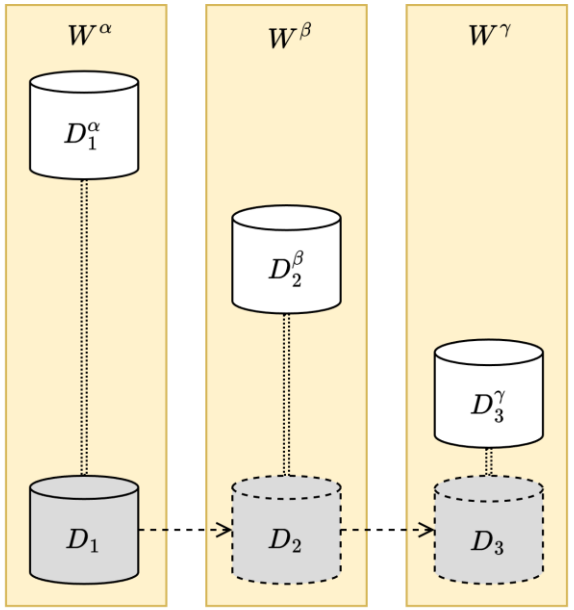
Experimental setting (I)

- Identify and download snapshots
- Upload to PostgreSQL as flat JSON (D_1, D_2, D_3)
- Extract corresponding workloads ($W^{\alpha}, W^{\beta}, W^{\gamma}$)
    - Collect queries
    - Cluster by tables involved
    - Exclude irrelevant clusters (aggregated frequency <1%)
    - Subcluster by projections and filters
    - Exclude irrelevent subclusters (aggregated frequency <0.5%)



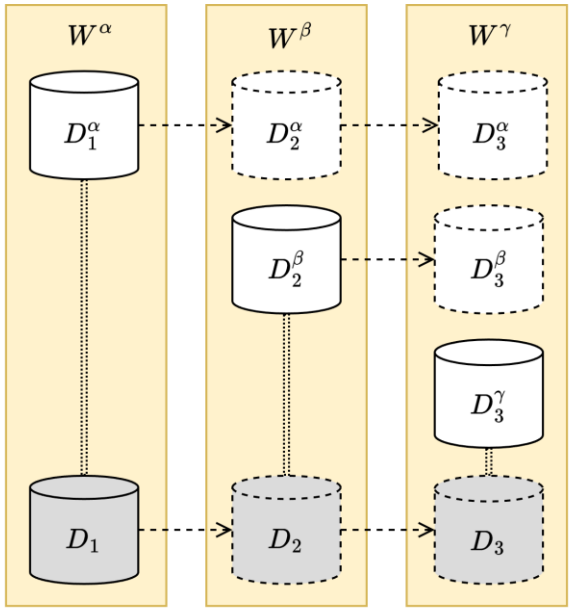
Experimental setting (I)

- Identify and download snapshots
- Upload to PostgreSQL as flat JSON (D_1, D_2, D_3)
- Extract corresponding workloads ($W^\alpha, W^\beta, W^\gamma$)
  - Collect queries
  - Cluster by tables involved
  - Exclude irrelevant clusters (aggregated frequency <1%)
  - Subcluster by projections and filters
  - Exclude irrelevent subclusters (aggregated frequency <0.5%)
- Create optimal design for workload ($D_1^\alpha, D_2^\beta, D_3^\gamma$)
  - Greedy iteration over queries by desc frequency to implement
    - Embedding of related instances in the main table
    - Vertical partitioning (i.e., projecting relevant attributes)
    - Horizontal partitioning (i.e., applying filters)
  - No data loss, no redundancies
  - Indexes created for all queries



Experimental setting (II)

- Evolve optimized schemas ($D_2^{\alpha}, D_3^{\alpha}, D_3^{\beta}$)
  - Preserve optimizations
  - Extended with concepts introduced in new release (one extra table per new concept)
  - **Crucial to put optimizations under the test of time**



Experimental setting (II)

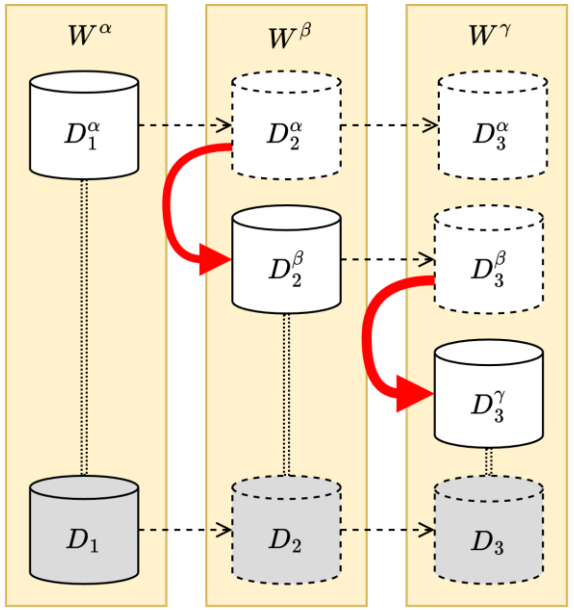
- Evolve optimized schemas ($D_2^{\alpha}, D_3^{\alpha}, D_3^{\beta}$)
  - Preserve optimizations
  - Extended with concepts introduced in new release (one extra table per new concept)
  - **Crucial to put optimizations under the test of time**

- Research question

  Given the previous optimizations (made at time 1 for W^{α}),
  will they hold at time 2 for W^{β} or
  will it be more convenient to migrate D_2^{α} to D_2^{β} ?

  $$execTime\left(W^{\beta}, D_2^{\alpha}\right) > execTime\left(W^{\beta}, D_2^{\beta}\right) + migrTime(D_2^{\alpha}, D_2^{\beta})$$

  (same question for W^{γ})



Experiments – Workload and storage

Workload statistics

- Frequency assumes uniform distribution in time
- Patterns by output cardinality

Work.	Queries per month	Query freq. (Hz)	N. query patterns by output 1 row (stable)	>1 rows (stable)	>1 rows (scaling)
W^{α}	48,337	0.02	2	3	-
W^{β}	1,352,498	0.57	6	3	-
W^{γ}	3,485,018	1.39	6	2	7

- Workload becomes increasily complex

Storage occupation

- Scales by cardinality of main table (PhotoObjAll)

Scale	D_1	D_1^{α}	D_2	D_2^{α}	D_2^{β}	D_3	D_3^{α}	D_3^{β}	D_3^{γ}
100K	683	704	849	875	853	885	892	882	878
200K	1365	1408	1600	1652	1607	1672	1685	1666	1665
500K	3412	3520	3852	3983	3869	4035	4065	4019	4028
1M	6824	7039	7607	7868	7640	7973	8033	7941	7965

- Storage intuitively increases with scale
- No significant impact of optimizations
  - No inter-table redundancies
  - Low footprint of intra-table redundancies

Experiments – Execution times

Setup

- 11K queries (first 1K discarded to minimize cold-start)
  - Random order, preserving frequency
  - Random values for selection predicates
- $avgQueryExecTime(W) = \sum_{q \in W} avgExecTime(q) * freq(q)$

Workload	Database	Scale 100K	200K	500K	1M
W^{α}	D_1 D_1^{α}	73.8 ± 7.2	71.3 ± 6.6	71.3 ± 6.0	77.9 ± 7.0
W^{β}	D_2 D_2^{α} D_2^{β}	27.7 ± 5.5	26.3 ± 5.1	25.5 ± 5.2	27.8 ± 5.3
W^{γ}	D_3 D_3^{α} D_3^{β} D_3^{γ}	145.9 ± 12.9	285.4 ± 17.4	714.5 ± 29.1	1125.2 ± 38.8

(times in ms)

Observations

- Workload changes significantly across releases

Experiments – Execution times

Setup

- 11K queries (first 1K discarded to minimize cold-start)
  - Random order, preserving frequency
  - Random values for selection predicates
- $avgQueryExecTime(W) = \sum_{q \in W} avgExecTime(q) * freq(q)$

Workload	Database	Scale 100K	200K	500K	1M
W^{α}	D_1	73.8 ± 7.2	71.3 ± 6.6	71.3 ± 6.0	77.9 ± 7.0
	D_1^{α}	7.6 ± 2.4	5.4 ± 2.3	5.4 ± 2.1	5.4 ± 2.2
W^{β}	D_2	27.7 ± 5.5	26.3 ± 5.1	25.5 ± 5.2	27.8 ± 5.3
	D_2^{α}				
	D_2^{β}	2.0 ± 2.8	1.6 ± 1.9	1.6 ± 2.3	1.0 ± 2.4
W^{γ}	D_3	145.9 ± 12.9	285.4 ± 17.4	714.5 ± 29.1	1125.2 ± 38.8
	D_3^{α}				
	D_3^{β}				
	D_3^{γ}	45.5 ± 13.5	60.7 ± 8.6	125.5 ± 13.4	240.6 ± 21.9

(times in ms)

Observations

- Workload changes significantly across releases
- Optimizations have a huge impact (3x-10x reduction)

Experiments – Execution times

Setup

- 11K queries (first 1K discarded to minimize cold-start)
  - Random order, preserving frequency
  - Random values for selection predicates
- $avgQueryExecTime(W) = \sum_{q \in W} avgExecTime(q) * freq(q)$

Workload	Database	Scale 100K	200K	500K	1M
W^{α}	D_1	73.8 ± 7.2	71.3 ± 6.6	71.3 ± 6.0	77.9 ± 7.0
	D_1^{α}	7.6 ± 2.4	5.4 ± 2.3	5.4 ± 2.1	5.4 ± 2.2
W^{β}	D_2	27.7 ± 5.5	26.3 ± 5.1	25.5 ± 5.2	27.8 ± 5.3
	D_2^{α}	63.7 ± 8.0	60.2 ± 8.4	60.2 ± 8.0	63.7 ± 8.4
	D_2^{β}	2.0 ± 2.8	1.6 ± 1.9	1.6 ± 2.3	1.0 ± 2.4
W^{γ}	D_3	145.9 ± 12.9	285.4 ± 17.4	714.5 ± 29.1	1125.2 ± 38.8
	D_3^{α}	571.2 ± 15.7	1124.4 ± 21.0	1546.4 ± 28.9	3249.2 ± 39.5
	D_3^{β}	269.7 ± 10.6	490.9 ± 14.2	1134.6 ± 24.5	2304.3 ± 33.5
	D_3^{γ}	45.5 ± 13.5	60.7 ± 8.6	125.5 ± 13.4	240.6 ± 21.9

(times in ms)

Observations

- Workload changes significantly across releases
- Optimizations have a huge impact (3x-10x reduction)
- Optimizations **don't** outlive the workload they were made for

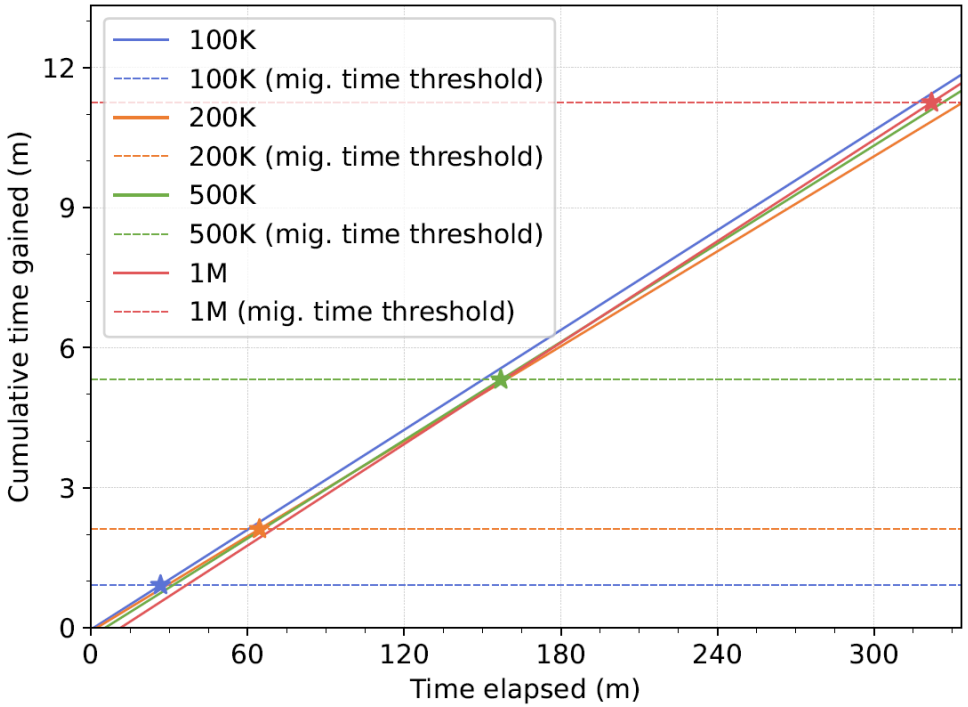
Experiments – Migration convenience (I)

Setup

- Migration queries ran 10 times (shown as dotted horizontal lines)
- $gainPerSec = \frac{avgQueryExecTime(W^{\beta}, D_2^{\alpha}) - avgQueryExecTime(W^{\beta}, D_2^{\beta})}{numQueriesPerSec}$

Observations

- Migration time is proportional to database size
- $cumGain$ **is not** affected by database size
  - Query results **don't** scale
- Migration convenience depends on database size
  - 25m to 5h in samples → convenient
  - >1month in full scale → possibly not convenient (if next month's workload differs significantly)



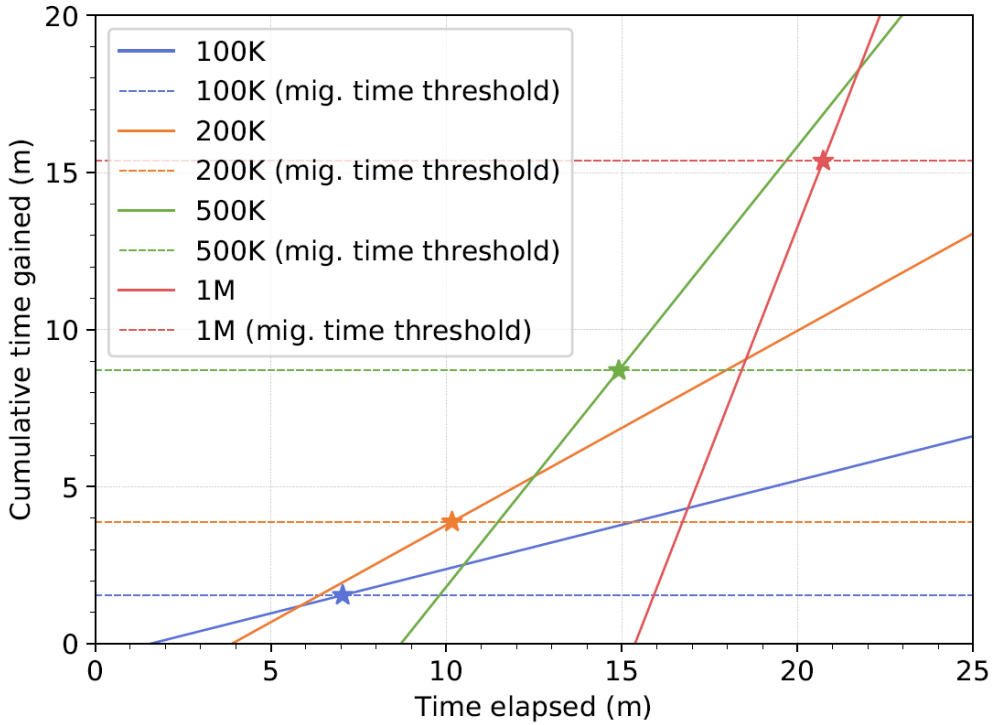
Experiments – Migration convenience (II)

Setup

- Migration queries ran 10 times (shown as dotted horizontal lines)
- $gainPerSec = \frac{avgQueryExecTime\left(W^{\gamma}, D_3^{\beta}\right) - avgQueryExecTime\left(W^{\gamma}, D_3^{\gamma}\right)}{numQueriesPerSec}$

Observations

- Migration time is proportional to database size
- $cumGain$ **is** affected by database size
  - Query results **do** scale
- Migration becomes convenient in just 6-7 minutes
  - Independently of the scale



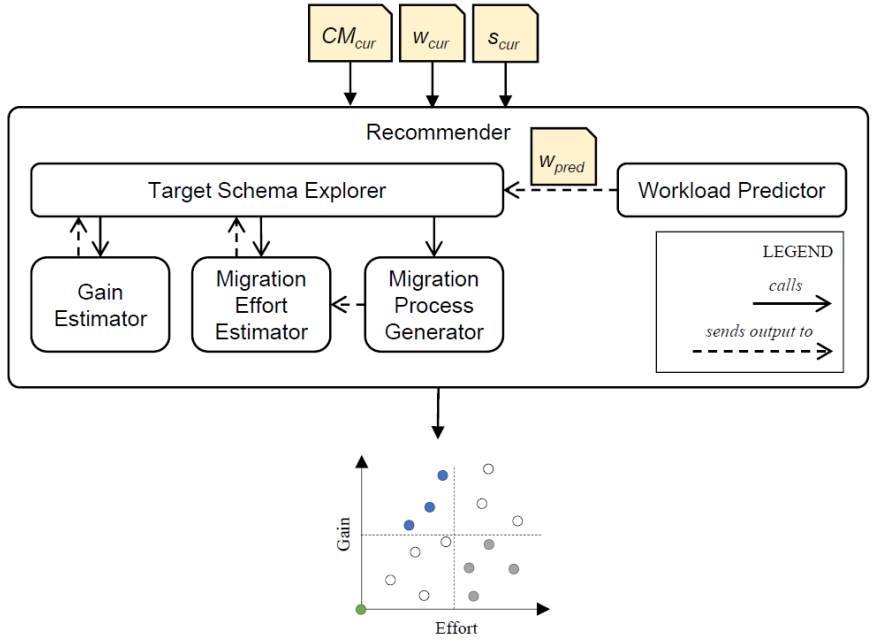
Refactoring framework

Outcomes

- Database optimization is neither a one-time nor an incremental activity
  - Some optimizations can become counterproductive in future stages
- Database migration can be particularly expensive, may not be worth the trouble
- Continuous evaluation of refactoring options is fundamental to guarantee maximum efficiency under evolving workloads

Our proposal

- A multi-objective optimization to continuously evaluate and recommend the refactoring of NoSQL databases



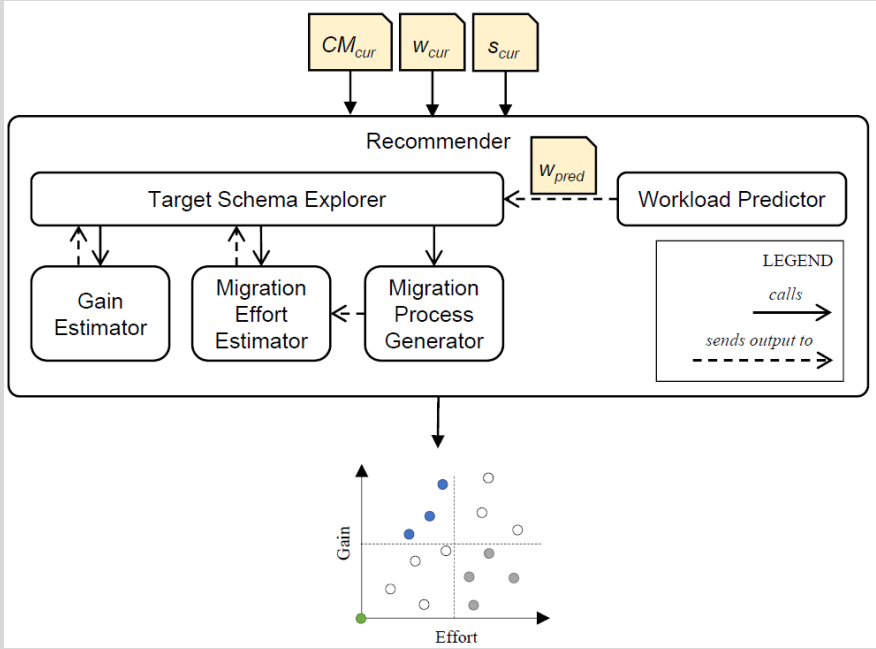
Refactoring framework

Human-in-the-loop automation

- System provides critical insights to make refactoring decisions with significant improvements
  - Based on objective criteria and a comprehensive coverage of possible alternatives
- DBAs/engineers contribute with their knowledge and exploit the system to make decisions

Main challenges

- Migration effort estimation
  - Existing KPIs are limited
  - Empirical evidence is not usable
  - Multiple factors
    - Design, execution, downtime, application update
- Workload prediction
  - Optimality of the new target schema may be lost by the time that the migration is completed



Optimizing migration sequence

A migration can be represented as a lattice, where

- Migration queries (*m*) create and populate the tables of the new DB
- There could be dependencies between migration queries
- Migration queries enable workload queries (*q*) to be rewritten in a more optimized way
- Migration and workload queries are respectively associated with a duration and a benefit

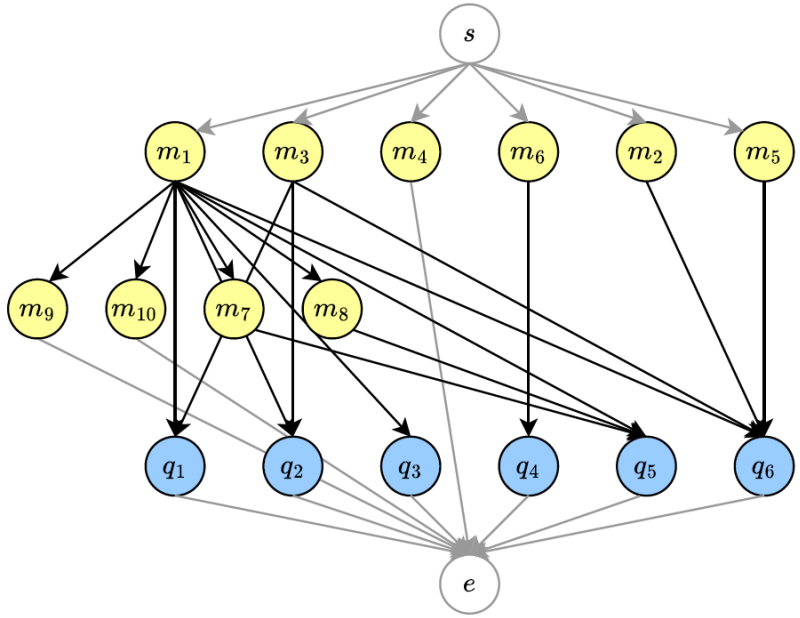
Assuming that

- Migration queries are executed in sequence

Goal

- Determine the plan (*P*) that maximizes the performance gain

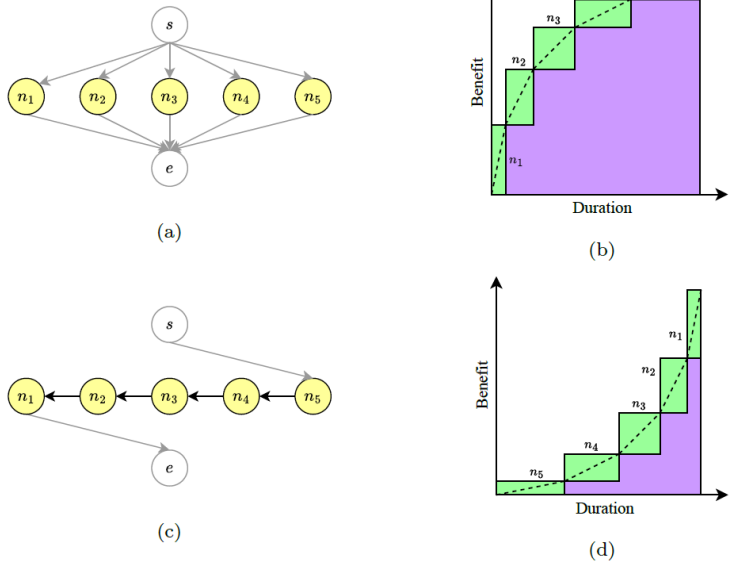
$$Ben(P) = \sum_{1 \leq i < |P|} Ben(P[i]) \cdot \sum_{i < j \leq |P|} Dur(P[j])$$



The logic

Independently of the sequence, the final benefit will be the same, but

- If we start by unlocking with low effort (duration) the rewritten workload queries with the highest gain (benefit), we maximize the potential of the migration
- The optimal solution is obtained by sorting nodes by decreasing benefit-duration ratio [WSPT]

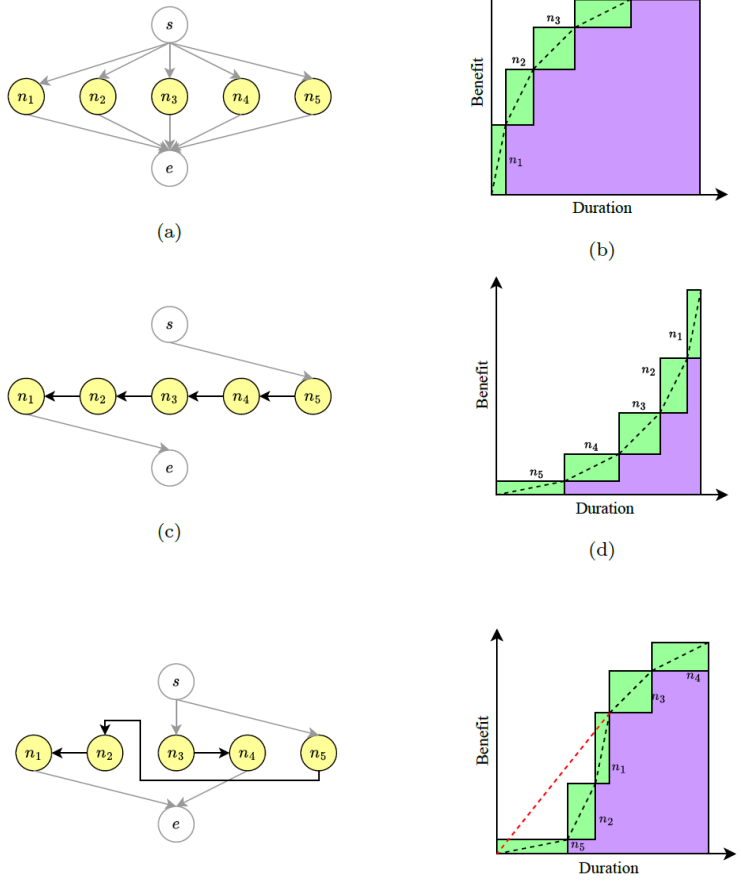


[WSPT] Smith, W. & Others Various optimizers for single-stage production. Naval Research Logistics Quarterly. 3, 59-66 (1956)

The logic

Independently of the sequence, the final benefit will be the same, but

- If we start by unlocking with low effort (duration) the rewritten workload queries with the highest gain (benefit), we maximize the potential of the migration
- The optimal solution is obtained by sorting nodes by decreasing benefit-duration ratio [WSPT] – only in the absence of precedence constraints!
  - We need to compare nodes based on a form of promised benefit-duration ratio
- The search space is M! in the worst case
  - We need a heuristics



[WSPT] Smith, W. & Others Various optimizers for single-stage production. Naval Research Logistics Quarterly. 3, 59-66 (1956)

Our heuristics

Formal background

$$ImmBen(m) = \sum_{q \in Q \cap succ(m)} Ben(q) \cdot Contrib(m, q)$$

$$PromBen(n_i) = ImmBen(n_i) + \sum_{n_j \in PromSucc(n_i)} PromBen(n_j)$$

$$PromDur(n_i) = Dur(n_i) + \sum_{n_j \in PromSucc(n_i)} PromDur(n_j)$$

$$PromSucc(m) = \bigcup_{m_i \in M \cap succ(m)} m_i : \frac{PromBen(m_i)}{PromDur(m_i)} \geq \frac{ImmBen(m)}{Dur(m)}$$

$$H(n) = \begin{cases} \dfrac{PromBen(n)}{PromDur(n)} & \text{if } n \in M \\ \text{sgn}(Ben(n)) \cdot \infty & \text{if } n \in Q \end{cases}$$

The solution is determined by sorting nodes by *H(n)*

Algorithm 1 GreedySearch(L)

Require: L: the precedence lattice.
Ensure: P: the migration plan.

```
P ← []
H ← GenerateHeuristics(L)   ▷ Launch an inverse topological sort of L
ready ← [s]                 ▷ s is the starting node of L
while ¬empty(ready) do
    bestNode ← argmax_{n∈ready} H[n]
    P.append(bestNode)
    enabledNodes ← {n ∈ succ(bestNode) | ∀n' ∈ pred(n) : n' ∈ P}
    ready ← (ready \ bestNode) ∪ enabledNodes
end while
return P
```

Outcomes

Tests were made on both SkyServer case study and synthetic examples

Results showed that

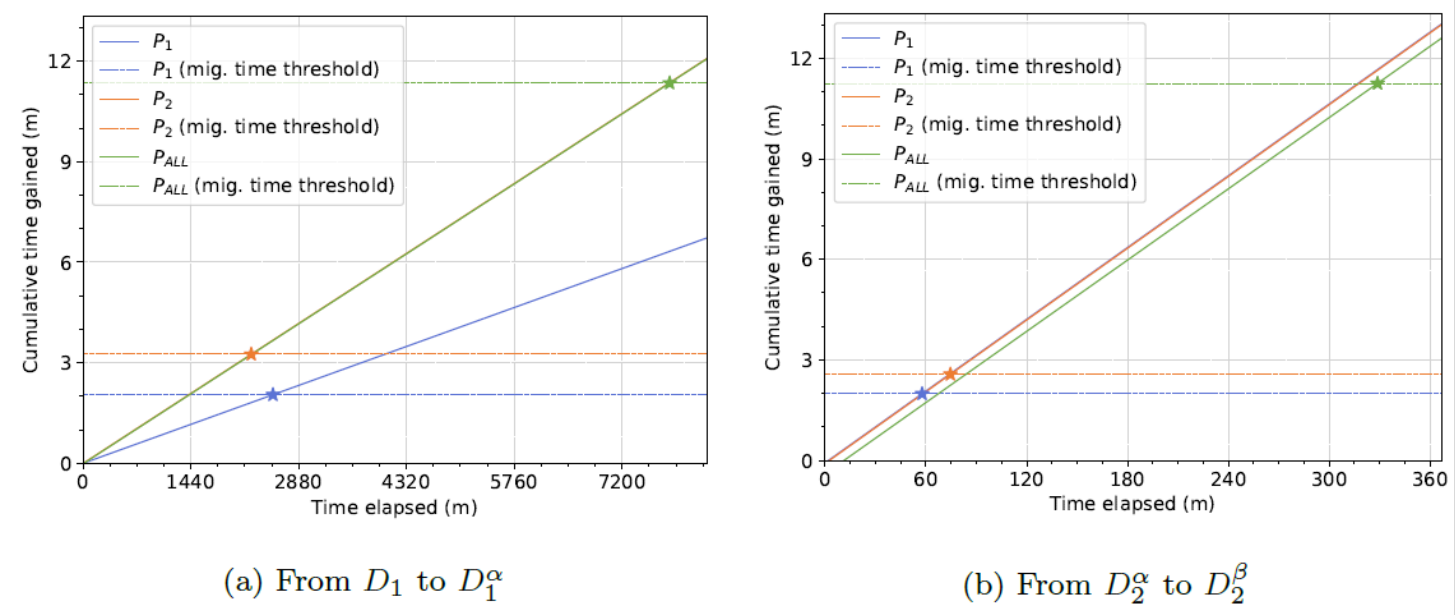
- The heuristics always succeeded in returning the optimal plan
- The heuristics performed several orders of magnitude better than exhaustive search (trivially)
- Simpler strategies (e.g., always start from the node with lower duration) could diverge significantly from the optimal plan

Outcomes

Tests were made on both SkyServer case study and synthetic examples

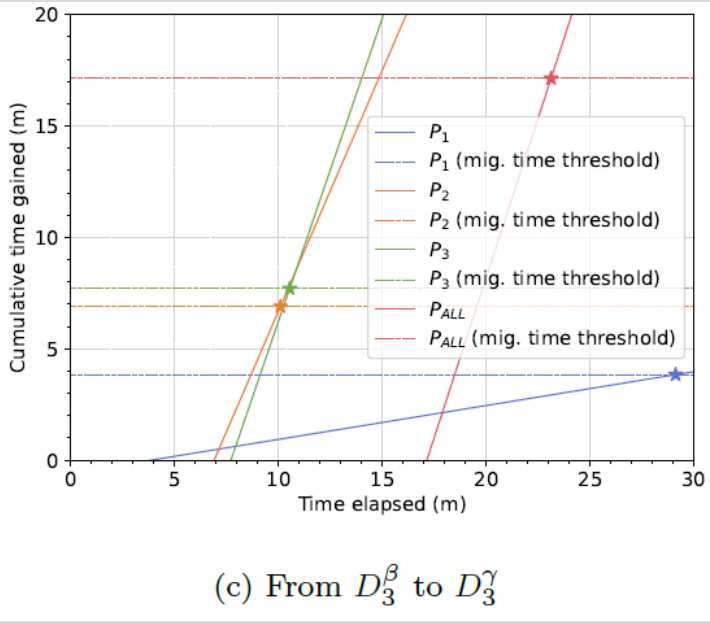
Results showed that

- In some cases, partial migrations are sufficient



(a) From D_1 to D_1^{α}

(b) From D_2^{α} to D_2^{β}

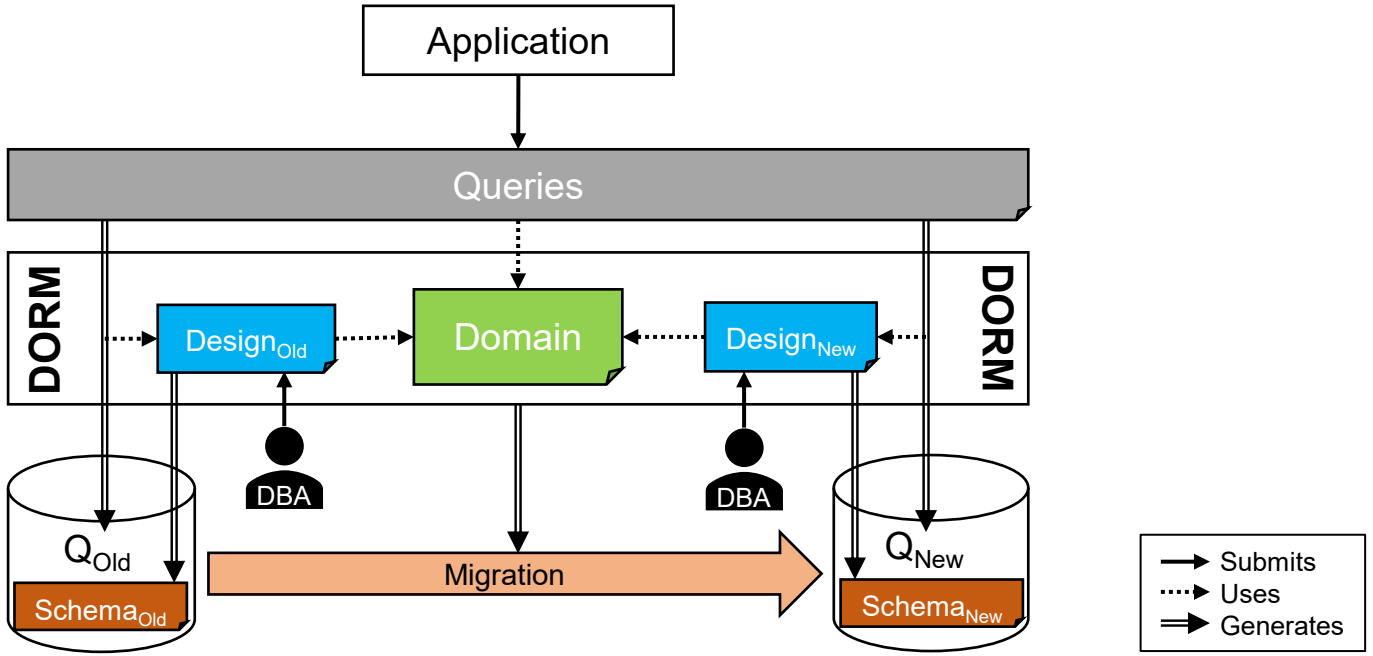


(c) From D_3^{β} to D_3^{γ}

DORM

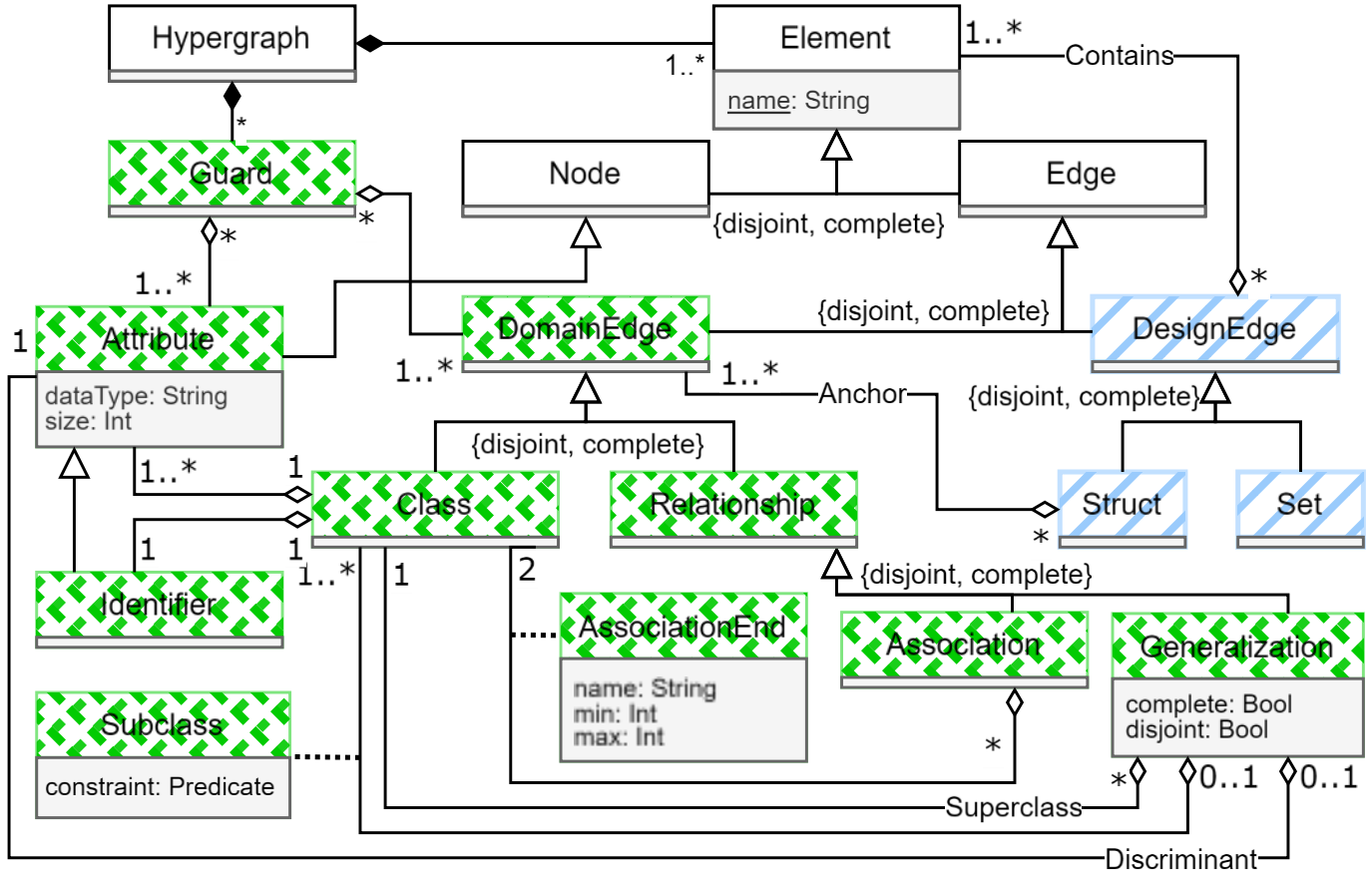
Dynamic Object-Relational Mapping (DORM)

- A tool to automate database migration
- https://github.com/alberto0723/DORM



DORM

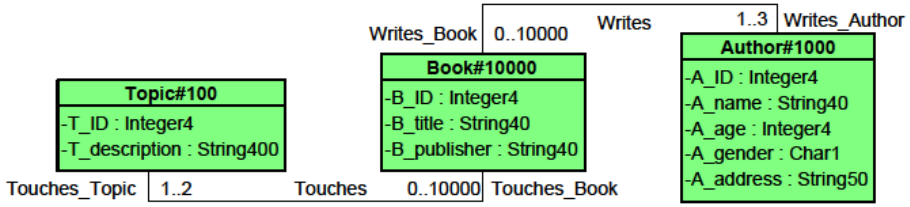
Metamodel to represent the domain (green) and the logical design (blue)

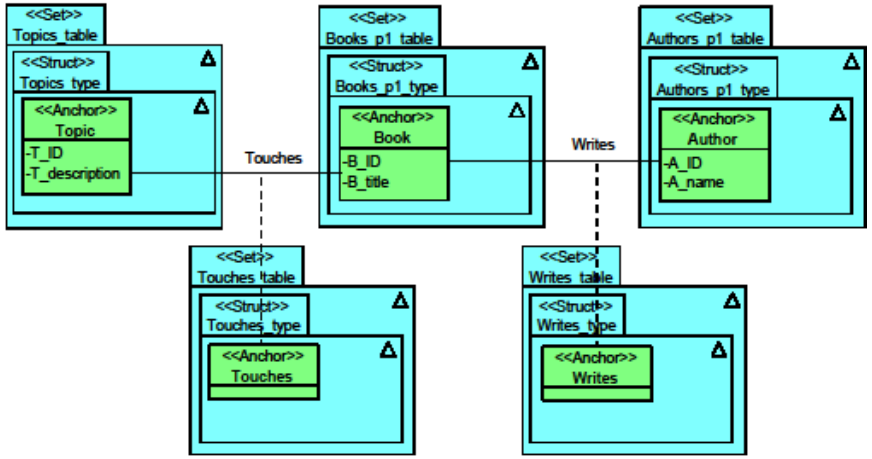


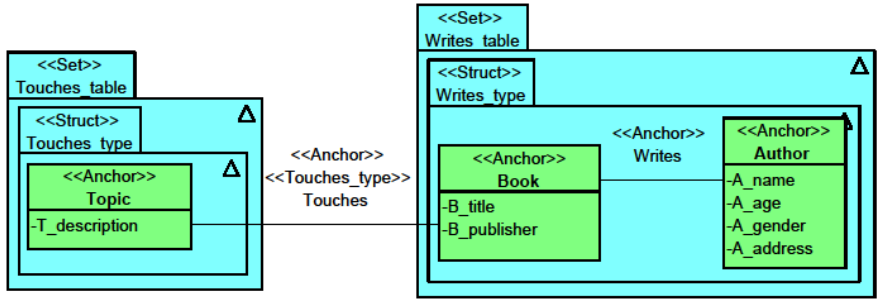
DORM

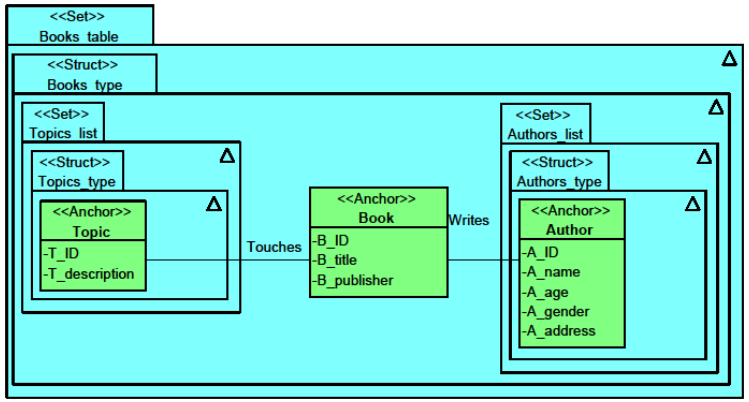
Supports several schema design choices

- Nesting, flattening, vertical and horizontal partitionings





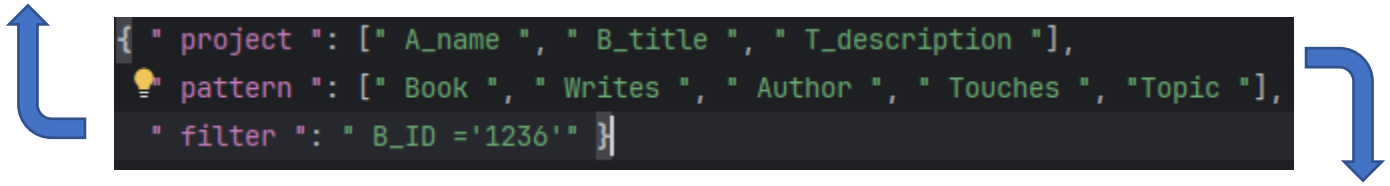




DORM

Supports query rewriting

```
SELECT t_1.A_name AS A_name, t_2 . B_title AS B_title , t_3.T_description AS T_description
FROM Authors_table t_1
  JOIN Writes_table t_5 ON t_1 . A_ID = t_5 . Writes_Author
  JOIN Books_table t_2 ON t_5 . Writes_Book = t_2 . B_ID
  JOIN Touches_table t_4 ON t_2 . B_ID = t_4 . Touches_Book
  JOIN Topics_table t_3 ON t_4 . Touches_Topic = t_3. T_ID
WHERE ( t_2 . B_ID ='1236 ');
```



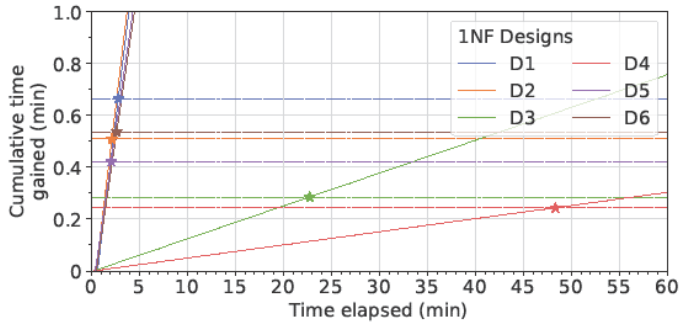
```
{ " project ": [" A_name ", " B_title ", " T_description "],
 " pattern ": [" Book ", " Writes ", " Author ", " Touches ", "Topic "],
  " filter ": " B_ID ='1236'" }
```

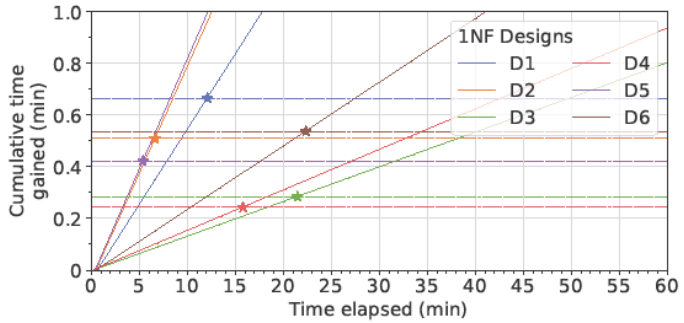
```
SELECT t_2->>'A_name ' AS A_name , t_1.value ->>'B_title ' AS B_title , t_3->>' T_description ' AS T_description
FROM Books_table t_1
  JOIN LATERAL jsonb_array_elements ( t_1 .value ->'Authors_list ') t_2 ON TRUE
  JOIN LATERAL jsonb_array_elements ( t_1 .value ->'Topics_list ') t_3 ON TRUE
WHERE ( t_1 .value ->>'B_ID '='1236 ');
```

DORM

Tested on SkyServer for further cost-benefit tradeoff analyses

1NF	Concept Representation								Workload	
Design	Association	PhotoObjAll	PhotoObj	PhotoPrimary	Photoz	SpecObjAll	SpecObj	Migration	May	June
Baseline	FK	Table			Table	Table		-	412	98
D1	FK	Horizontal P.	Horizontal P.	Vertical P.	Join Mat.	Table		39972	48	72
D2	Table	Horizontal P.	Horizontal P.	Vertical P.	Join Mat.	Table		30568	44	61
D3	FK	Table			Join Mat.	Horizontal P.	Vertical P.	17096	396	92
D4	Table	Table			Join Mat.	Horizontal P.	Vertical P.	14645	405	91
D5	FK++	Horizontal P.	Horizontal P.	Table	Join Mat.	Horizontal P.	Vertical P.	25387	109	60
D6	Table	Horizontal P.	Horizontal P.	Table	Join Mat.	Horizontal P.	Vertical P.	32269	114	87





Conclusion

Takeaways

- Domains and workloads evolve
- (NoSQL) schema optimizations make huge improvements in the short run
- (NoSQL) schema optimizations can become a liability in the long run
- Database refactoring may not always be worth executing
- Continuous database refactoring is always worth evaluating

What's next

- Considering additional strategies for schema optimizations
- Incorporating "future" workloads for a more comprehensive test-of-time
- *[DOING]* Improving state-of-the-art on NoSQL schema design
- Defining a method for estimating migration efforts
- *[DOING]* Extending automation in DORM

Issues with workload-driven DB design

1. What if the domain and/or the workload change?
2. **What is the best design for a workload?**
   a. **Which criteria define a design as the *best* one?**
   b. How do we obtain it?

Measuring the cost of a schema design

Matteo Francia, Enrico Gallinucci, Matteo Golfarelli, Alessandra Lumini

CM²: a Cloud Monetary Cost Model for Document-Based NoSQL Databases

~~Submitted to VLDB J.~~

Motivation

Paving the way to a cloud-based recommender system

- Several approaches have already been proposed
  - Atzeni, deLima, Chen, Abello, …, DaMoOp
- Cloud environments mostly overlooked
- Before recommending, we need an approach to measure costs

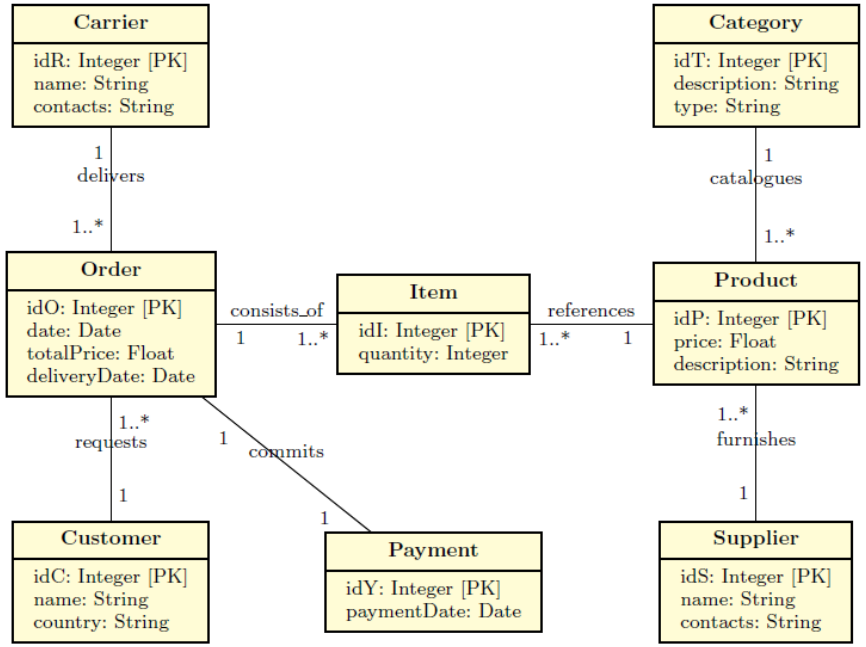
The good and bad of cloud-based databases

- Serverless model → no need to consider hardware infrastructure
- Paid service → everything (storage and compute) can be reduced to a single metric ($)
- Proprietary implementation → internal mechanisms (and pricing) are a black box

Case study

Domain

- Onlinestore
- Represented as a DAG



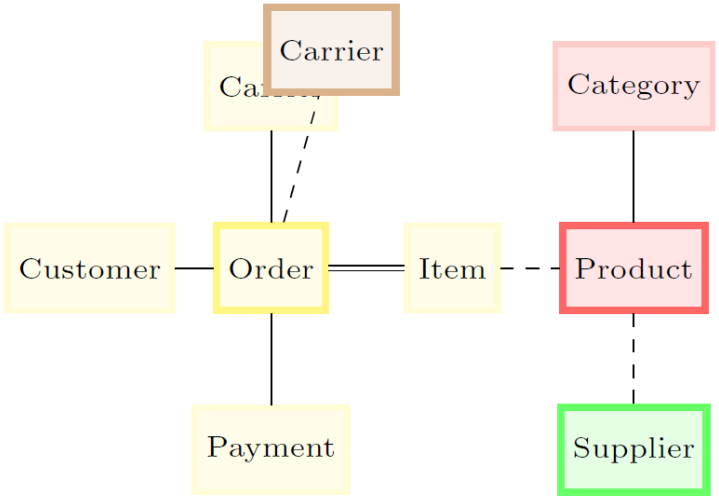
Queries

- Both read and write
- Conjunctive selection predicates, varying selectivity
- No cycles

Logical design of collections

A relationship between two entities can be implemented via

- Referencing (using FKs; 3NF)
- Nesting: denormalizing the instances of an entity inside the other
  - Embedding: nesting inside the *to-one* entity (e.g. Order embeds Items) → creates a list of Items
  - Flattening: nesting inside the *to-many* entity (e.g. Order flattens Customers) → replicates Customers



Collections: [Or ≺ (Cr, It, Cu, Py)], [Pr ≺ Ct],[Su],[Cr]

Customer ≺ Order	Order ≺ Customer
{ "idC": 1, "name": "Alice", "orders": [{ "idO": 101, "date": "01/01/2024", "idR": 1}, { "idO": 102, "date": "02/01/2024", "idR": 2 }] }	{ "idO": 101, "date": "01/01/2024", "idC": 1, "name": "Alice", "idR": 1 } { "idO": 102, "date": "02/01/2024", "idC": 1, "name": "Alice", "idR": 2 }

Query plans

A query may be executed in multiple ways

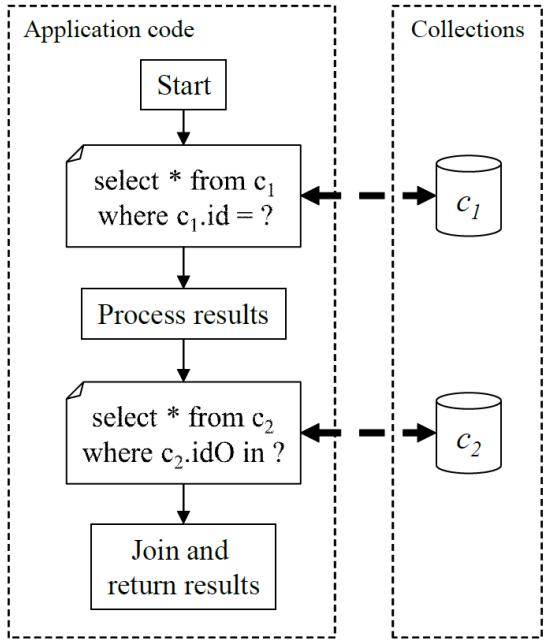
- Lots of works on relational databases [6]
- Due to redundancies, the same query can be answered by involving different collections
- Cloud-based NoSQL databases do not support joins → application-side join

Query plan (assumed as given)

- A sequence of access plans, $qp = [ap_1, \ldots, api]$
- Associated with a frequency, $freq(q)$

Access plan

- A tuple $ap = (c, SP, jp, ix)$
  - c is the collection
  - sp is the set of selection predicates in the form $(attr, nv)$
  - jp is the join predicate
  - ix is the index used

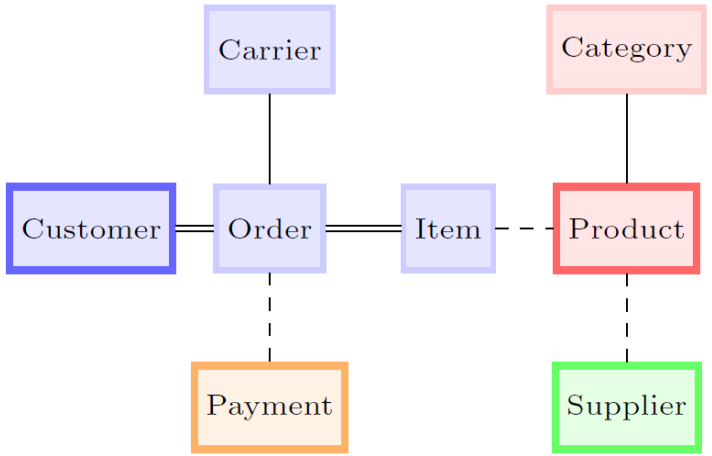


Query plans - example

Query: retrieve the details of an Order given a date, including information about the related Customer, Payment, and Items

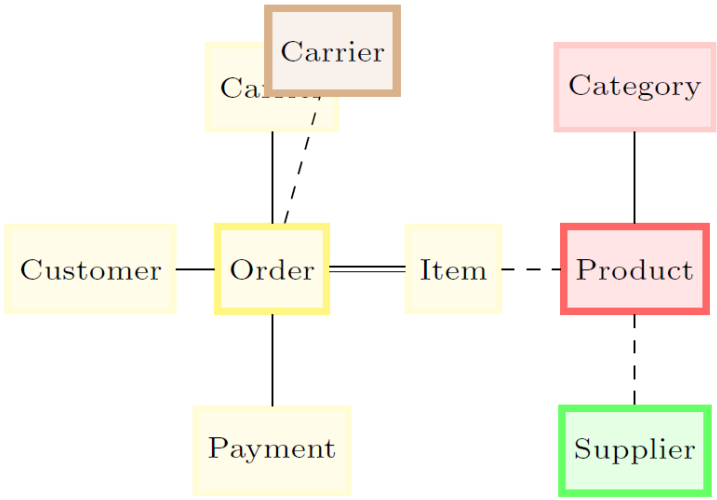
$ap_1.c = [Cu \prec Or \prec (Cr, It)]$
$ap_1.SP = \{(Or.date, 1)\}$
$ap_1.jp = \emptyset, ap_1.ix = [Or.date]$

$ap_2.c = [Py], ap_2.SP = \emptyset,$
$ap_2.jp = (1, Py.idY, Or.idY)$
$ap_2.ix = [Py.idY]$



Collections: [Cu ≺ Or ≺ (Cr, It)], [Pr≺ Ct],[Su],[Py]

$ap_1.c = [Or \prec (Cr, It, Cu, Py)]$
$ap_1.SP = \{(Or.date, 1)\}$
$ap_1.jp = \emptyset, ap_1.ix = [Or.date]$



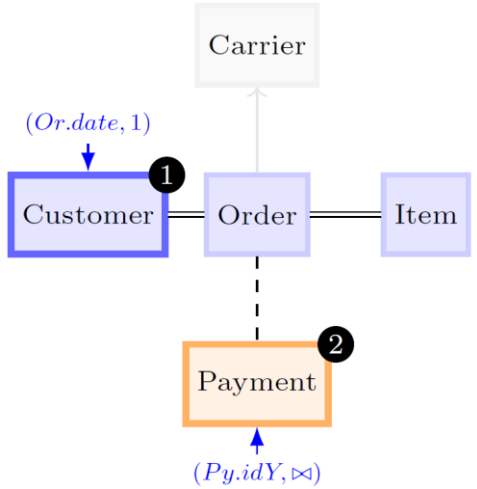
Collections: [Or ≺ (Cr, It, Cu, Py)], [Pr ≺ Ct],[Su],[Cr]

Query plans - example

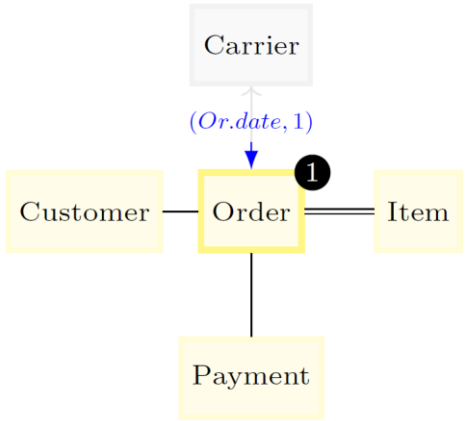
Query: retrieve the details of an Order given a date, including information about the related Customer, Payment, and Items

$ap_1.c = [Cu \prec Or \prec (Cr, It)]$
$ap_1.SP = \{(Or.date, 1)\}$
$ap_1.jp = \emptyset, ap_1.ix = [Or.date]$

$ap_2.c = [Py], ap_2.SP = \emptyset,$
$ap_2.jp = (1, Py.idY, Or.idY)$
$ap_2.ix = [Py.idY]$



$ap_1.c = [Or \prec (Cr, It, Cu, Py)]$
$ap_1.SP = \{(Or.date, 1)\}$
$ap_1.jp = \emptyset, ap_1.ix = [Or.date]$



The cost model

Input

- A set of logical schemas of collections
- A set of query plans, associated with index usage and a monthly frequency

Assumptions

- Multi-collection queries are split into single-collection queries
  - Join predicates automatically inferred
- Document writes (i.e., insert/update/delete) are allowed only by referencing PK values
  - Write query plans are composed of a single access plan
  - Any read operation needed to retrieve the PK values must be accounted for as read query plans

The monetary costs is determined by:

- Estimating the number of accessed documents and index entries
- Estimating the space occupation (in bytes) of the collections
  - Including the respective indexes, automatically determined in accordance with the query plans
- Translating the above into monetary cost

The cost model

The main formula

$$\kappa(C, RQP, WQP) = \sum_{\substack{ap \in qp, \\ qp \in RQP}} \kappa^R(ap) \cdot freq(qp) + \sum_{\substack{ap \in qp, \\ qp \in WQP}} \kappa^W(ap) \cdot freq(qp) + \sum_{c \in C} \kappa^S(c)$$

The components

$$\kappa^R(ap) = max(\ \$^{QR}, k^{IR}(\lambda^{ent}(ap), \omega^{ent}(ap.ix)) + k^{DR}(\lambda^{doc}(ap), \omega^{doc}(ap.c))\ )$$

$$\kappa^W(ap) = max(\ \$^{QW}, k^{DW}(\lambda^{doc}(ap), \omega^{doc}(ap.C))\ )$$

$$\kappa^S(c) = k^S(\omega^{col}(c) + \sum_{ix \in c.IX} \omega^{ix}(ix))$$

Symbol	Description
$\kappa(C, RQP, WQP)$	Monetary cost of storing the collections in C and querying them through the read and write query workloads RQP and WQP
$\kappa^R(ap)$	Monet. cost of executing a read access plan
$\kappa^W(ap)$	Monet. cost of executing a write access plan
$\kappa^S(c)$	Monet. cost of storing a collection c
$\lambda^{ent}(qp)$	N. index entries accessed by a query plan
$\lambda^{ent}(ap)$	N. index entries accessed by an access plan
$\lambda^{doc}(qp)$	N. documents accessed by a query plan
$\lambda^{doc}(ap)$	N. documents accessed by an access plan
$\omega^{col}(c)$	N. bytes required for the whole collection c
$\omega^{doc}(c)$	N. bytes required for a single document of c
$\omega^{ix}(ix)$	N. bytes required for the whole index ix
$\omega^{ent}(ix)$	N. bytes required for a single entry of ix

Pricing component	Description	Sample values
$bytesPerType(t)$	Number of bytes per data type $t \in$ [boolean, char, date, number]	[1, 1, 8, 8]
$docOH$	Number of additional bytes allocated for each document	100
$\$^{QR}$	Minimum of dollars paid per read operation	$3 \cdot 10^{-7}$
$\$^{QW}$	Minimum of dollars paid per write operation	$1 \cdot 10^{-6}$
$\$^{IR}$	Dollars paid per byte of index entry read	$1 \cdot 10^{-16}$
$\$^{DR}$	Dollars paid per byte of document read	$1 \cdot 10^{-12}$
$\$^{DW}$	Dollars paid per byte of document written	$1 \cdot 10^{-8}$
$\$^{S}$	Dollars paid per GB stored, monthly	$2.5 \cdot 10^{-10}$
$k^{IR}(n, s)$	Monetary cost of reading n index entries of size s	$n \cdot s \cdot \$^{IR}$
$k^{DR}(n, s)$	Monetary cost of reading n documents of size s	$n \cdot s \cdot \$^{DR}$
$k^{DW}(n, s)$	Monetary cost of writing/updating n documents of size s	$n \cdot s \cdot \$^{DW}$
$k^{S}(n)$	Monetary cost of storing n bytes	$n \cdot \$^{S}$

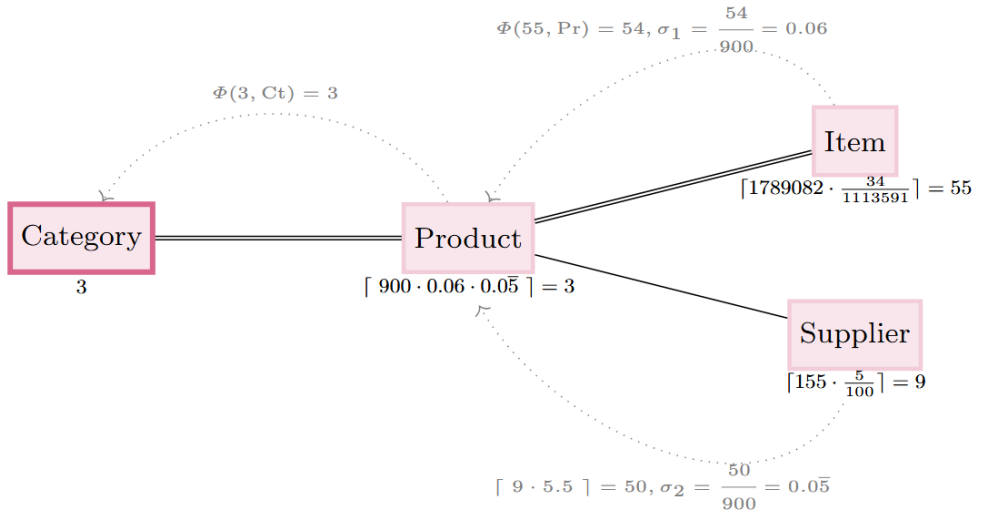
The challenges

1) Traversing relationships in a collection

- Goal
  - Estimate the number of accessed instances in the presence of nesting and filters)
- Principles
  - In the to-many direction, multiply by the average cardinality of the relationship
  - In the to-one direction, rely on Cardenas

$$\Phi(n,e) = \left\lceil \; card(e) \cdot \left(1 - \left(1 - \frac{1}{card(e)}\right)^{n}\right) \; \right\rceil$$

- Recursive, bottom-up strategy
  - Get cardinality of children entities
  - Apply filter selectivity
  - Propagate selectivity to parent entities



The challenges

2) Estimating index cardinality

- Goal
  - Quantify the number of entries (cardinality) and pointers per entry
- Principles
  - If single-attribute, the cardinality corresponds to the number of distinct values
  - If multi-attribute, the cardinality is bounded by
    - The product of the cardinalities of the attributes
    - The maximum cardinality of the entities of the indexed attributes (or in the path between them)

$$card(ix) = min(\prod_{a \in ix.A} card(a), \max_{e \in E_{mst}} (card(e)))$$

  - The number of pointers per entry is determined by the traversing function

The challenges

3) Estimating the number of accessed documents and index entries

- Goal
  - Determine cost of accessing documents and indexes
- Principles
  - Apply filter and join selectivities
  - Not all documents read may be returned
- Variables
  - λ^{ent} (number of index entries read)
  - λ^{doc} (number of documents read)
  - λ^{res} (number of documents returned)

Algorithm 3 $nDocIx$

Require: $qp = [ap_1, \ldots, ap_l]$: the access plans of a query plan.

Ensure: $\lambda^{ent}(qp)$: the estimated total number index entries accessed by qp; $\lambda^{doc}(qp)$: the estimated total number of documents accessed by qp.

1: **for** $ap_i \in qp$ **do**
2: $\quad SP \leftarrow ap_i.SP \cup getJoinSp(qp, ap_i)$
3: $\quad ISP = \{sp \in SP : sp.a \in ap_i.ix.A\}$ ▷ Indexed SPs
4: $\quad \lambda^{ent}(ap_i) \leftarrow \frac{\prod_{sp \in ISP} sp.nv}{\prod_{sp \in ISP} \text{card}(sp.a)} \cdot card(ap_i.ix)$
5: $\quad \lambda^{doc}(ap_i) \leftarrow filterInst(ap_i.c, ap_i.c.p, ISP)$
6: $\quad \lambda^{res}(ap_i) \leftarrow filterInst(ap_i.c, ap_i.c.p, SP)$
7: $\lambda^{ent}(qp) \leftarrow \sum_{ap \in qp} \lambda^{ent}(ap)$
8: $\lambda^{doc}(qp) \leftarrow \sum_{ap \in qp} \lambda^{doc}(ap)$
9: **return** $\lambda^{ent}(qp), \lambda^{doc}(qp)$

The challenges

4) Estimating storage size

- Goal
  - Determine space occupation of collections and indexes
- Principles
  - Must take into account
    - Size of attiribute names
    - Attribute types
    - Number (and size) of nested instances
    - Per-document overhead
    - Index modeling style (relational- or document-like)

$$attSize(a, n) = |name(a)| \cdot bytesPerType(\text{“string”}) + n \cdot avgLen(a) \cdot bytesPerType(type(a))$$

Algorithm 5 *getDocSize*

Require: c: a collection; e: an entity, $numInst$: the number of instances of e in c.
Ensure: the estimated storage occupation in bytes.

$docSize \leftarrow \sum_{a \in e} attSize(a, 1)$
for $e' \in c.T : e \prec_r e'$ **do**
  $docSize \leftarrow docSize + getDocSize(c, e', relInst(1, e, e', r))$
if $c.p = e$ **then** ▷ Only for the primary entity
  $docSize \leftarrow docSize + docOH$
return $docSize \cdot numInst$

Algorithm 6 *getIxSpace*

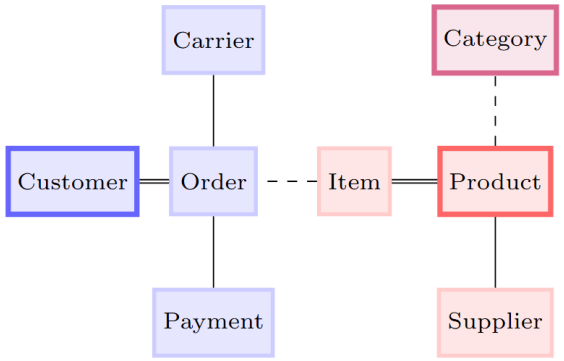
Require: c: a collection; ix: an index of c, $ixModel$: the data model of the index (relational or document-based).
Ensure: $\omega^{ent}(ix)$: the estimated storage occupation in bytes of an index entry; $\omega^{ix}(ix)$: the estimated storage occupation in bytes of the whole index.

$ev \leftarrow avgLen(c.p.key) \cdot bytesPerType(type(c.p.key)) \cdot ppe(ix)$ ▷ Size of an entry value
if $ixModel$ is document-based **then** ▷ Add size of indexed attributes to each entry
  $\omega^{ent}(ix) \leftarrow \sum_{a \in ix.A} attSize(a, 1) + ev$
else
  $\omega^{ent}(ix) \leftarrow ev$
$\omega^{ix}(ix) \leftarrow \omega^{ent}(ix) \cdot card(ix)$
return $\omega^{ent}(ix), \omega^{ix}(ix)$

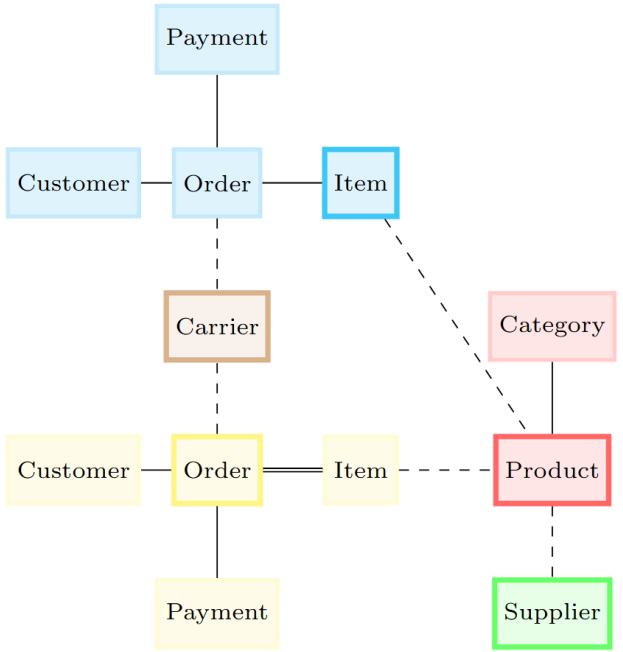
Experiments

Two reference solutions on Azure CosmosDB

- Diverse in terms of modeling choices and redundancies
- Customer-Order-Item implemented in every way
  - Full embedding: Cu > Or > It
  - Full flattening: It > Or > Cu
  - Mixed: Or > (Cu, It)



(a) Representation of C^{α}, composed of:
c_1^{α} =[Cu $\prec$ Or $\prec$ (Cr, Py)], c_2^{α} =[Pr$\prec$ (It, Su)], c_3^{α} = [Ct]



(b) Representation of C^{β}, composed of:
$c_1^{\beta} = [\text{Or} \prec (\text{Cu}, \text{It}, \text{Py})]$, $c_2^{\beta} = [\text{It} \prec \text{Or} \prec (\text{Cu}, \text{Py})]$, $c_3^{\beta} = [\text{Cr}]$, $c_4^{\beta} = [\text{Pr} \prec \text{Ct}]$, $c_5^{\beta} = [\text{Su}]$

Experiments

Workload of 18 (mostly read) queries

- Diverse in terms of span and selectivity

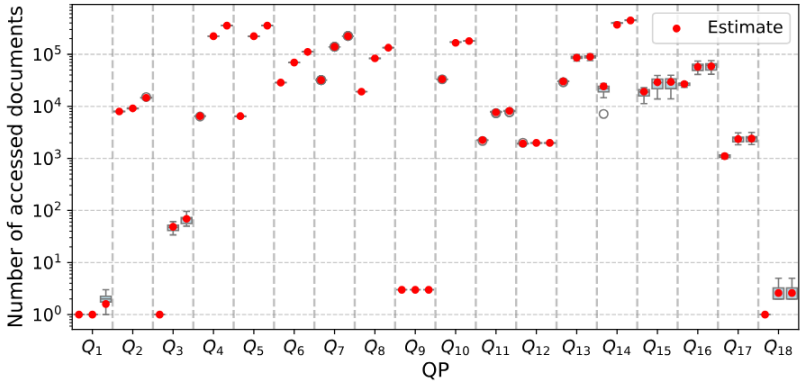
Query	Accessed entities	Filters	Collections accessed in workload W^{α}	$W^{\beta 1}$	$W^{\beta 2}$
Q_1 Q_2	Or	(Or.idO, 1) (Or.date, 3)	c_1^{α}	c_1^{β}	c_2^{β}
Q_3 Q_4 Q_5	Cu, Or, Cr	(Cu.idC, 1) (Cu.country, 10) (Cu.country, 10), (Cr.name, 10)	c_1^{α}	c_1^{β}, c_3^{β}	c_2^{β}, c_3^{β}
Q_6 Q_7 Q_8	Cr, Or, Cu	(Cr.idR, 1) (Cr.name, 2) (Cr.name, 2), (Cu.country, 30)	c_1^{α}	c_3^{β}, c_1^{β}	c_3^{β}, c_2^{β}
Q_9 Q_{10} Q_{11}	(all)	(It.idI, 1) (It.quantity, 10) (It.quantity, 10), (Cu.country, 2)	c_2^{α}, c_3^{α}, c_1^{α}	c_1^{β}, c_4^{β}, c_5^{β}	c_2^{β}, c_4^{β}, c_5^{β}
Q_{12} Q_{13} Q_{14}	(all)	(Pr.idP, 1) (Pr.price, 10) (Pr.price, 50), (Su.name, 10)	c_2^{α}, c_3^{α}, c_1^{α}	c_4^{β}, c_5^{β}, c_1^{β}	c_4^{β}, c_5^{β}, c_2^{β}
Q_{15} Q_{16} Q_{17}	(all)	(Ct.idT, 1) (Ct.description, 2) (Ct.description, 2), (Cu.country, 2)	c_3^{α}, c_2^{α}, c_1^{α}	c_4^{β}, c_5^{β}, c_1^{β}	c_4^{β}, c_5^{β}, c_2^{β}
Q_{18}	Or	(Or.idO, 1)	c_1^{α}	c_1^{β}, c_2^{β}	c_1^{β}, c_2^{β}

Experiments

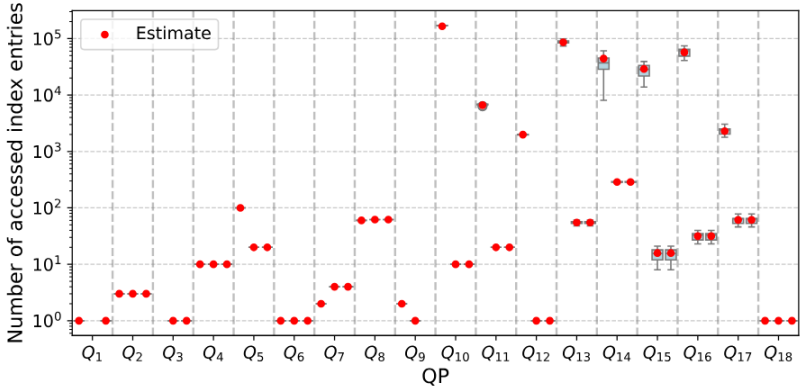
Comparison between estimated (red dots) and real (boxplots) numbers of accessed documents (a) and index entries (b)

- (a) APE = 1.68% ± 3.51%, R^2 = 0.9985
- (b) APE = 0.82% ± 2.58%, R^2 = 0.9981

- The uniform data distribution within the benchmark naturally leads to more stable observed costs



(a)



(b)

Experiments

Accuracy of storage estimates

- Statistics on small collections are not given
- Errors mostly observed on the smallest collections
  - Possibly due to collection-level overhead
  - Difference amounts to a few KBs
- Low precision of Azure's APIs limits the investigation on indexes
  - Occupation of indexes is only at collection-level, not at index-level
- Monetary cost
  - Est: 0,73 $/month
  - Real: 0,70 $/month
  - Compatible with GB-level APE

Coll.	Documents			Indexes		
	Est.	Actual	APE	Est.	Actual	APE
c_1^α	187 MB	173 MB	8.2%	58.9 MB	65.2 MB	9.8%
c_2^α	69.2 MB	67.8 MB	2.0%	29.7 MB	38.2 MB	22.1%
c_3^α	39.5 KB	-	-	2.07 KB	3.07 KB	32.8%
c_1^β	913 MB	908 MB	0.6%	129 MB	121 MB	6.6%
c_2^β	1.39 GB	1.29 GB	7.5%	162 MB	148 MB	9.3%
c_3^β	10.4 KB	-	-	811 B	2.05 KB	60.4%
c_4^β	617 KB	786 KB	21.6%	21.5 KB	14.3 KB	49.6%
c_5^β	107 KB	131 KB	18.6%	3.31 KB	15.4 KB	78.4%
TOT	2.56 GB	2.44 GB	4.8%	379 MB	372 MB	1.9%

Experiments

Azure's pricing model is based on the consumption of RU (Request Units)

- RU normalize the cost of all database operations; cost is determined by RU/s
- RU consumption is primarily based on the amount of moved bytes; empirical approximations:
  - ρ_{DR} = 30 KB/RU
  - ρ_{IR} = 300 KB/RU
  - ρ_{DW} = 60 B/RU

Pricing comp.	Short description	Value
$docOH$	Additional bytes per doc	1250
$\$^{QR}$	Min. $ per read op.	$3 \cdot \rho^{RU} = 7.5 \cdot 10^{-7}$ $
$\$^{QW}$	Min. $ per write op.	$6 \cdot \rho^{RU} = 1.5 \cdot 10^{-6}$ $
$\$^{IR}$	$ per byte of ix entry read	$\rho^{RU}/\rho^{IR} = 8.33 \cdot 10^{-17}$ $
$\$^{DR}$	$ per byte of doc read	$\rho^{RU}/\rho^{DR} = 8.33 \cdot 10^{-12}$ $
$\$^{DW}$	$ per byte of doc written	$\rho^{RU}/\rho^{DW} = 4.55 \cdot 10^{-9}$ $

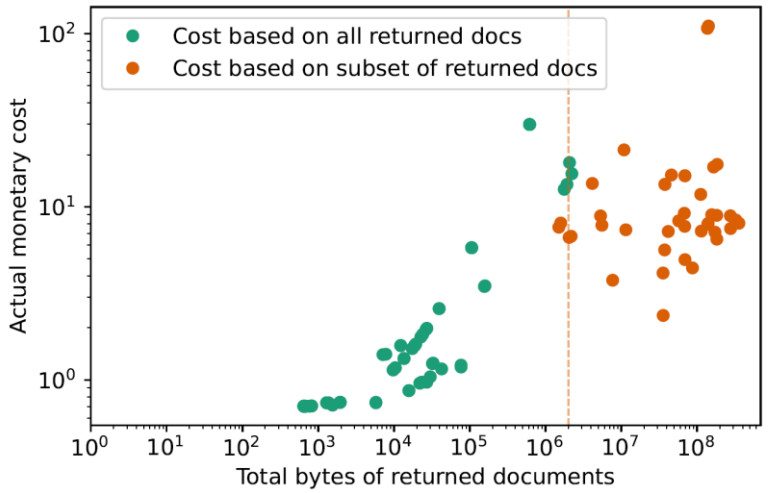
Experiments

Azure's pricing model is based on the consumption of RU (Request Units)

- RU normalize the cost of all database operations; cost is determined by RU/s
- RU consumption is primarily based on the amount of moved bytes; empirical approximations:
  - ρ_{DR} = 30 KB/RU
  - ρ_{IR} = 300 KB/RU
  - ρ_{DW} = 60 B/RU

Problem

- Correlation between RU (i.e., costs) and moved bytes disappears on expensive queries…



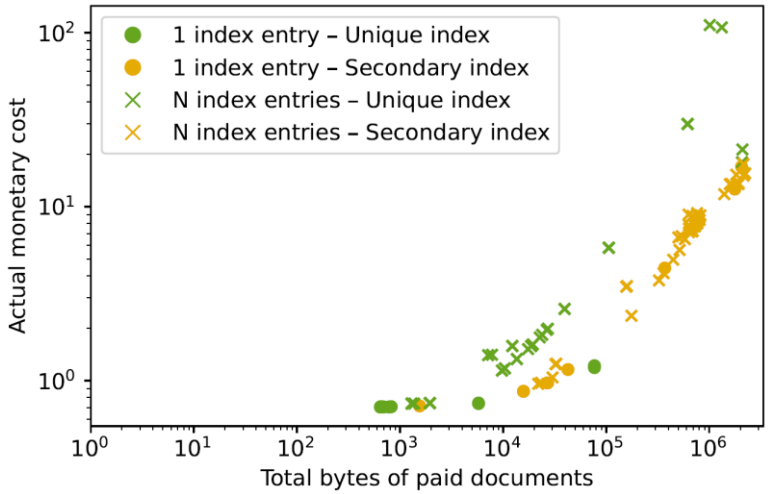
Experiments

Azure's pricing model is based on the consumption of RU (Request Units)

- RU normalize the cost of all database operations; cost is determined by RU/s
- RU consumption is primarily based on the amount of moved bytes; empirical approximations:
  - ρ_{DR} = 30 KB/RU
  - ρ_{IR} = 300 KB/RU
  - ρ_{DW} = 60 B/RU

Problem

- Correlation between RU (i.e., costs) and moved bytes disappears on expensive queries…
- …because, after a given threshold, the number of documents *charged* by Azure is different from the number of documents *returned*
  - Also notice a distinction between index access types

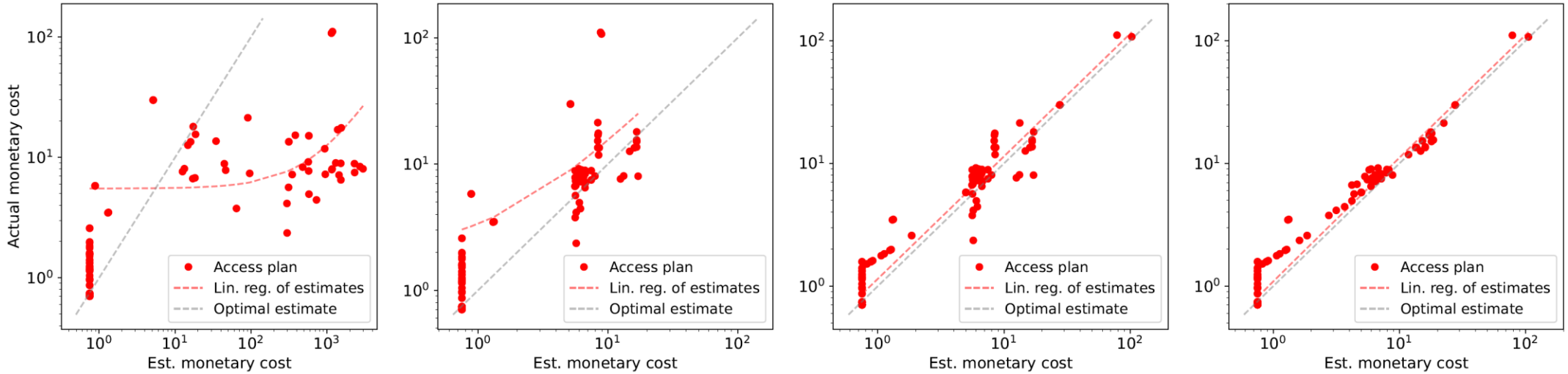


Experiments

Solution: just tune to cost model!

- V_0: basic version
- V_1: V_0 + it caps # of charged documents to 2MB
- V_2: V_1 + it differentiates RU consumption on index usage
- V_{oracle}: V_2 but assumes to know exactly the # of charged docs

Cost model version	Mean APE ± var.	Pearson
V_0 (Figure 16.a)	$3,303.71\% \pm 5,280.21\%$	0.2807
V_1 (Figure 16.b)	$33.65\% \pm 8.31\%$	0.4072
V_2 (Figure 16.c)	$24.94\% \pm 4.97\%$	0.9766
V_{oracle} (Figure 16.d)	$19.45\% \pm 2.48\%$	0.9834



(a) V_0 (b) V_1 (c) V_2 (d) V_{oracle}

Experiments

Is CM^2 useful to compare alternative modeling solutions?

- 4 modeling alternatives, 3 workload variations

DB	Workload	Workload variation	Monthly actual costs ($)				Monthly estimated cost ($)			
			Storage	Read	Write	Total	Storage	Read	Write	Total
C^{α}	W^{α}	$freq(Q*) = 1M$	0.09	447.51	23.06	470.66	0.09	372.16	24.07	396.32
$C^{\beta 1}$	$W^{\beta 1}$		0.26	294.72	3.91	298.88	0.26	270.22	3.42	273.90
$C^{\beta 2}$	$W^{\beta 2}$		0.36	292.37	5.81	298.54	0.39	261.17	6.45	268.01
C^{β}	W^{β}		0.62	283.50	9.72	**293.83**	0.65	258.35	9.87	**268.86**

- C^{α} is more "normalized" than C^{β} variations
  - Higher reading costs in the former
  - Not much difference between the latter

Experiments

Is CM^2 useful to compare alternative modeling solutions?

- 4 modeling alternatives, 3 workload variations

DB	Workload	Workload variation	Monthly actual costs ($)				Monthly estimated cost ($)			
			Storage	Read	Write	Total	Storage	Read	Write	Total
C^{α}	W^{α}	$freq(Q*) = 1M$	0.09	447.51	23.06	470.66	0.09	372.16	24.07	396.32
$C^{\beta 1}$	$W^{\beta 1}$		0.26	294.72	3.91	298.88	0.26	270.22	3.42	273.90
$C^{\beta 2}$	$W^{\beta 2}$		0.36	292.37	5.81	298.54	0.39	261.17	6.45	268.01
C^{β}	W^{β}		0.62	283.50	9.72	**293.83**	0.65	258.35	9.87	**268.86**
C^{α}	W^{α}	$freq(Q_3) = 1B$	0.09	1,189.27	23.06	**1,212.42**	0.09	1,121.41	24.07	**1,145.57**
$C^{\beta 1}$	$W^{\beta 1}$		0.26	3,546.47	3.91	3,550.63	0.26	2,517.97	3.42	2,521.65
$C^{\beta 2}$	$W^{\beta 2}$		0.36	3,731.43	5.81	3,737.60	0.39	2,508.92	6.45	2,515.76
C^{β}	W^{β}		0.62	3,535.24	9.72	3,545.57	0.65	2,506.10	9.87	2,516.61

- Q_3 favors a modeling present in C^{α}
  - Increasing its frequency, makes C^{α} more convenient

Experiments

Is CM^2 useful to compare alternative modeling solutions?

- 4 modeling alternatives, 3 workload variations

DB	Workload	Workload variation	Monthly actual costs ($)				Monthly estimated cost ($)			
			Storage	Read	Write	Total	Storage	Read	Write	Total
C^{α}	W^{α}	$freq(Q*) = 1M$	0.09	447.51	23.06	470.66	0.09	372.16	24.07	396.32
$C^{\beta 1}$	$W^{\beta 1}$		0.26	294.72	3.91	298.88	0.26	270.22	3.42	273.90
$C^{\beta 2}$	$W^{\beta 2}$		0.36	292.37	5.81	298.54	0.39	261.17	6.45	268.01
C^{β}	W^{β}		0.62	283.50	9.72	**293.83**	0.65	258.35	9.87	**268.86**
C^{α}	W^{α}	$freq(Q_3) = 1B$	0.09	1,189.27	23.06	**1,212.42**	0.09	1,121.41	24.07	**1,145.57**
$C^{\beta 1}$	$W^{\beta 1}$		0.26	3,546.47	3.91	3,550.63	0.26	2,517.97	3.42	2,521.65
$C^{\beta 2}$	$W^{\beta 2}$		0.36	3,731.43	5.81	3,737.60	0.39	2,508.92	6.45	2,515.76
C^{β}	W^{β}		0.62	3,535.24	9.72	3,545.57	0.65	2,506.10	9.87	2,516.61
C^{α}	W^{α}	$freq(Q_{18}) = 100M$	0.09	447.51	2,306.00	2,753.60	0.09	372.16	2,407.08	2,779.33
$C^{\beta 1}$	$W^{\beta 1}$		0.26	294.72	390.50	**685.48**	0.26	270.22	341.67	**612.15**
$C^{\beta 2}$	$W^{\beta 2}$		0.36	292.37	581.00	873.73	0.39	261.17	645.00	906.56
C^{β}	W^{β}		0.62	284.88	971.50	1,255.61	0.65	258.35	986.67	1,245.66

- Q_{18} is the writing query
  - Increasing its frequency favors solutions with lower redundancy

Experiments

Is CM^2 useful to compare alternative modeling solutions?

- 4 modeling alternatives, 3 workload variations

DB	Workload	Workload variation	Monthly actual costs ($)				Monthly estimated cost ($)			
			Storage	Read	Write	Total	Storage	Read	Write	Total
C^{α}	W^{α}	$freq(Q*) = 1M$	0.09	447.51	23.06	470.66	0.09	372.16	24.07	396.32
$C^{\beta 1}$	$W^{\beta 1}$		0.26	294.72	3.91	298.88	0.26	270.22	3.42	273.90
$C^{\beta 2}$	$W^{\beta 2}$		0.36	292.37	5.81	298.54	0.39	261.17	6.45	268.01
C^{β}	W^{β}		0.62	283.50	9.72	**293.83**	0.65	258.35	9.87	**268.86**
C^{α}	W^{α}	$freq(Q_3) = 1B$	0.09	1,189.27	23.06	**1,212.42**	0.09	1,121.41	24.07	**1,145.57**
$C^{\beta 1}$	$W^{\beta 1}$		0.26	3,546.47	3.91	3,550.63	0.26	2,517.97	3.42	2,521.65
$C^{\beta 2}$	$W^{\beta 2}$		0.36	3,731.43	5.81	3,737.60	0.39	2,508.92	6.45	2,515.76
C^{β}	W^{β}		0.62	3,535.24	9.72	3,545.57	0.65	2,506.10	9.87	2,516.61
C^{α}	W^{α}	$freq(Q_{18}) = 100M$	0.09	447.51	2,306.00	2,753.60	0.09	372.16	2,407.08	2,779.33
$C^{\beta 1}$	$W^{\beta 1}$		0.26	294.72	390.50	**685.48**	0.26	270.22	341.67	**612.15**
$C^{\beta 2}$	$W^{\beta 2}$		0.36	292.37	581.00	873.73	0.39	261.17	645.00	906.56
C^{β}	W^{β}		0.62	284.88	971.50	1,255.61	0.65	258.35	986.67	1,245.66

Answer: yes!

Issues with workload-driven DB design

1. What if the domain and/or the workload change?
2. **What is the best design for a workload?**
   a. Which criteria define a design as the *best* one?
   b. How do we obtain it?

Exploring schema design alternatives (and recommending one)

Work in progress

The essence of the problem

Take any cost model that enables the ranking of solutions

How do we want find the *best* one?

- The search space is huge

Approaches typically rely on heuristics

- Common-sense rules are used to explore only few alternatives
- The optimal solution may not be found

Quantifying the search space

First, let’s fix some boundaries/rules

- Acyclic queries and workload
- No horizontal/vertical partitioning
- No optional relationships
- Every entity and relationship is requested by some query
- No redundancy of root entities
  - E.g., can’t have two Customer collections, even if internally different
- Each relationship can be modeled in one of three ways
  - Referencing, Embedding, Flattening

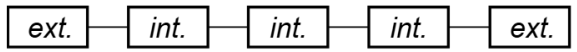
Quantifying the search space

Let C_e denote all possible implementations for a single collection, rooted on entity e

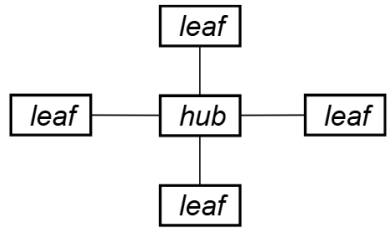
- $\dot{C}_e = C_e + \emptyset$

The cardinality of C_e depends on the topology of the domain

- Two reference topologies: / linear (simplest) and $*$ radial (most complex)

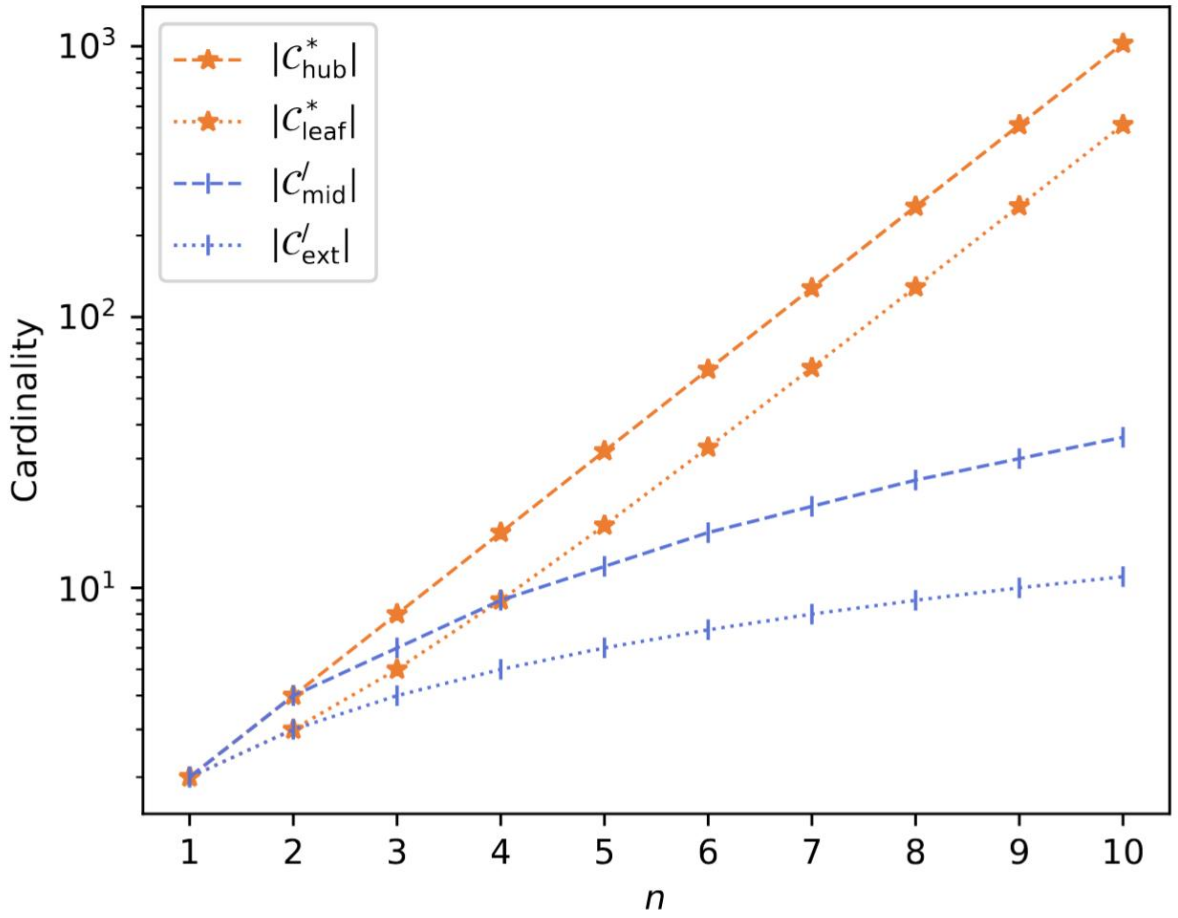


$$|\mathcal{C}_e^{/}| = (maxDepth(e) + 1) \cdot (n - maxDepth(e) + 1)$$



$$|\mathcal{C}_e^{*}| = \begin{cases} 2^n & \text{if } e \text{ is the hub} \\ 2^{n-1} + 1 & \text{if } e \text{ is a leaf} \end{cases}$$

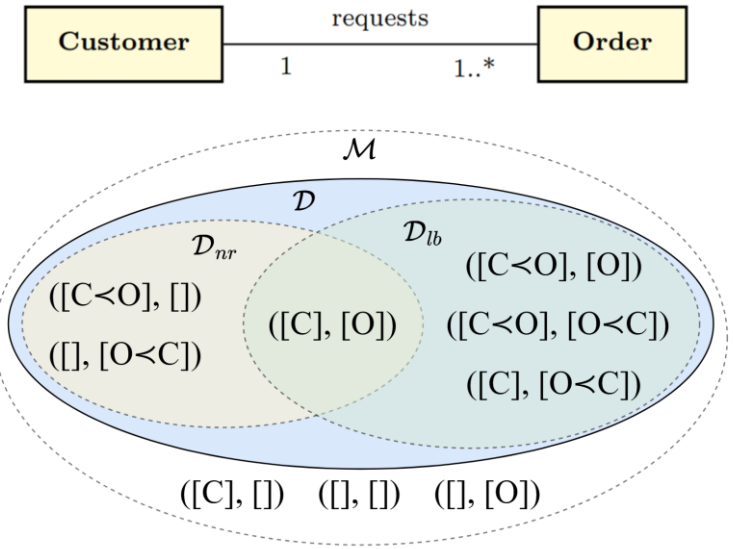
Quantifying the search space



Quantifying the search space

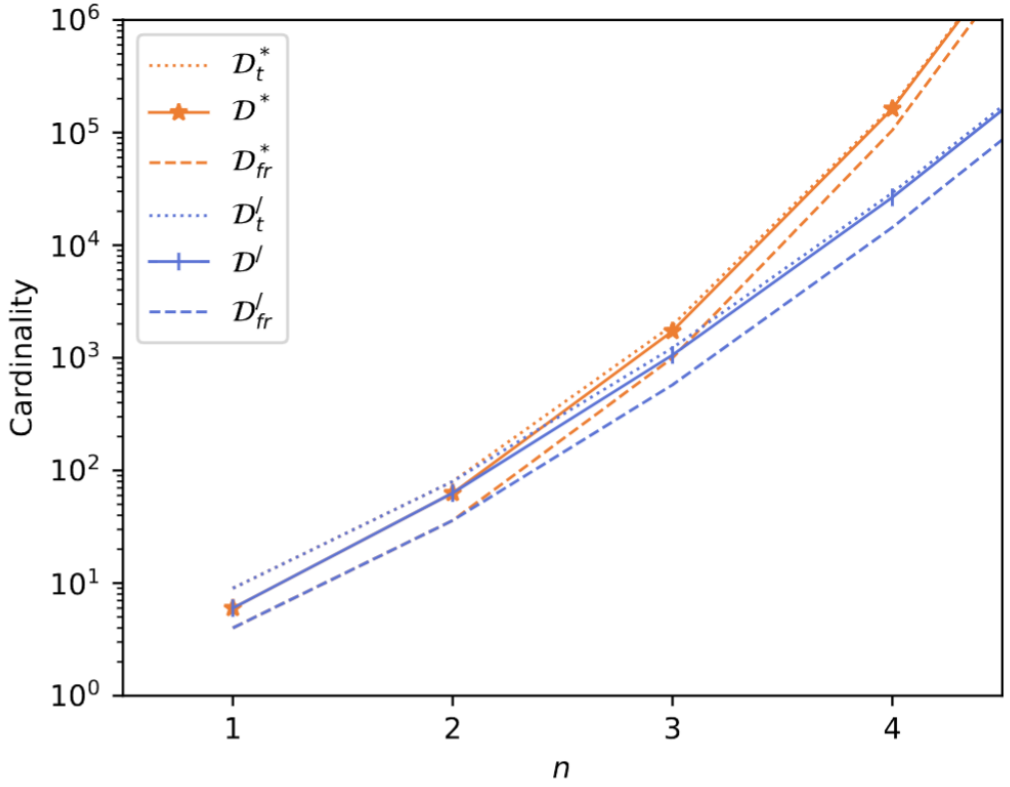
Interesting sets are

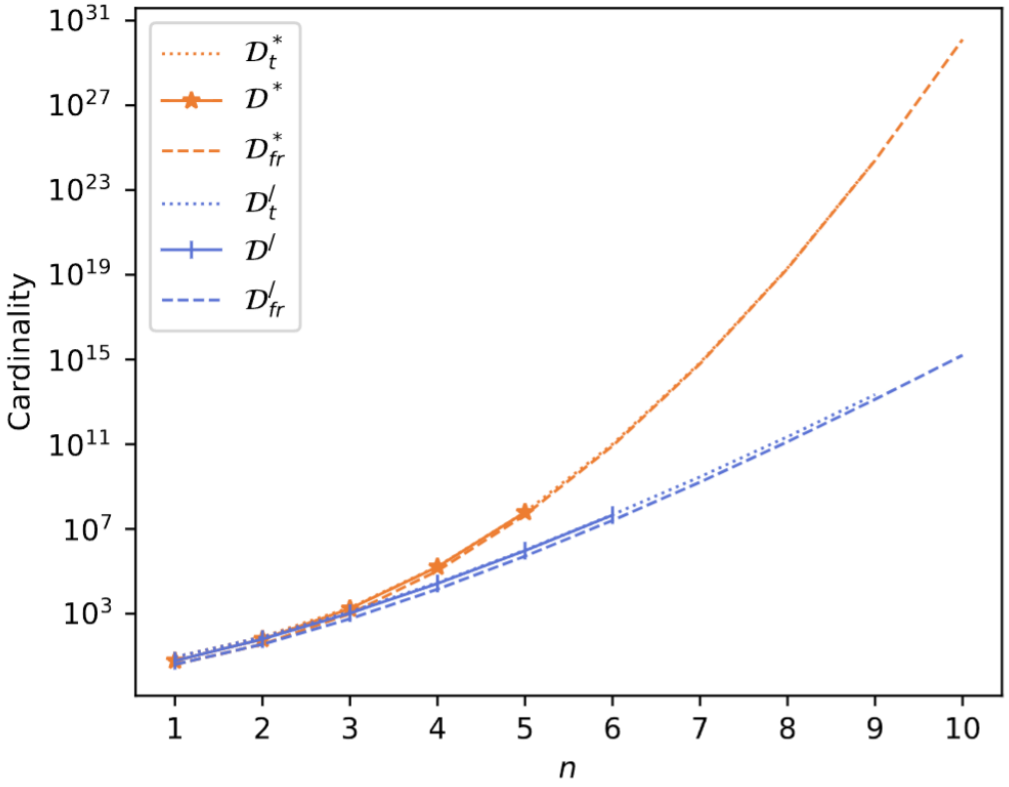
- M : universe of solutions (upper bound to D)
  - $M \leq \prod_{e \in E} |\dot{C}_e|$
- D : covering solutions
- D_{nr} : non-redundant solutions (lower bound to D)
  - $D_{nr} \leq \{\text{ref}, \text{emb}, \text{flt}\}^{n-1}$
- D_{lb} : solutions where each entity is root in a collection (lower bound to D)
  - $D_{lb} = \prod_{e \in E} |C_e|$



Set	Description	Linear topology Formula	Linear topology Compl.	Radial topology Formula	Radial topology Compl.
$\mathcal{D}_{nr}$	Non-redundant database schemas	$Fib(2 \cdot n + 2)$	$O(2.618^n)$	$2^n + n \cdot 2^{n-1}$	$O(2^n)$
$\mathcal{D}_{lb}$	Lower bound to database schemas with redundancies	$(n+1)!^2$	$O(n^n)$	$(2^n + 2)^n$	$O(n^n)$
$\mathcal{M}$	Upper bound to database schemas with redundancies	$\prod_{i=1}^{n+2} (i \cdot (n+2-i)+1)$	$O(n^n)$	$(2^n + 4)^n + (2^{n-1} + 2)^n$	$O(n^n)$

Quantifying the search space





Exhaustive exploration

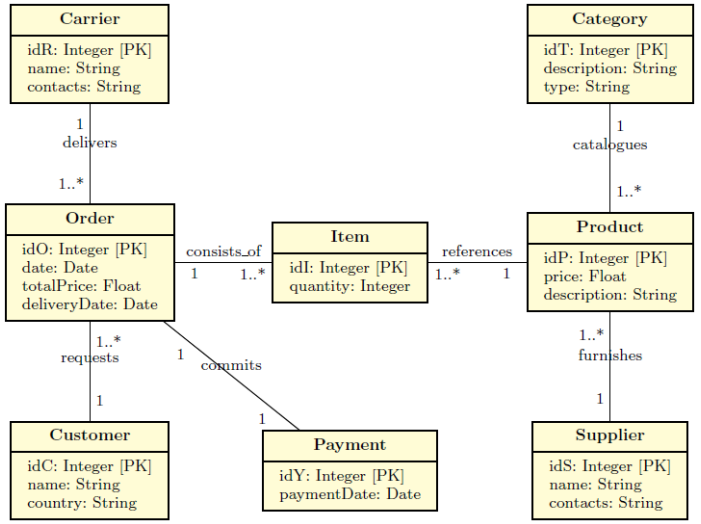
The complexity is super-exponential

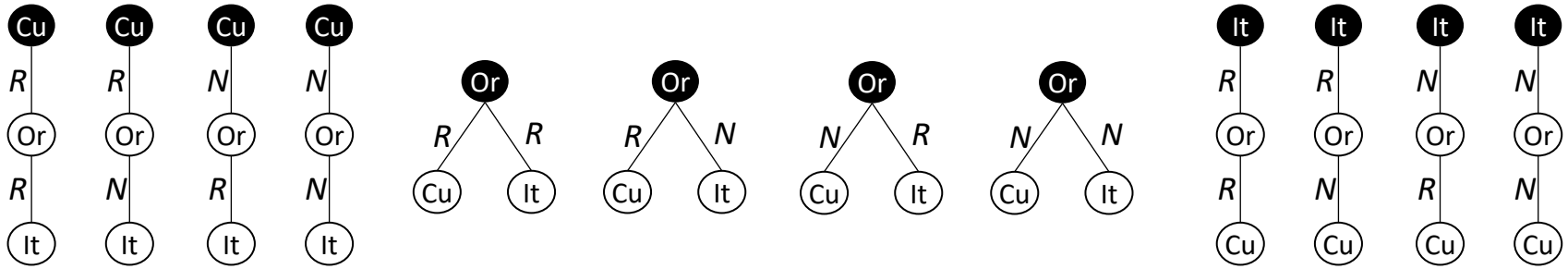
Still, is an exhaustive exploration feasible?

- In general, no
- With a limited n, maybe
- Excluding "useless" solutions, maybe
- What about a cost-model-based approach?

Exhaustive exploration

1) Generate annotated trees for each query
   - In annotated trees
     - Nodes/Edges are the Entities/Relationships involved
     - The root is *entry point* of the query
     - Edges are annotated with the logical modeling of the rel, i.e., either *R (referencing)* or *N (nesting)*
       - Whether N implies embedding of flattening is determined by the direction from the root
   - Consider *q* =
```
SELECT *
FROM Order, Customer, Item
WHERE Order.idO = 123 (AND joins)
```

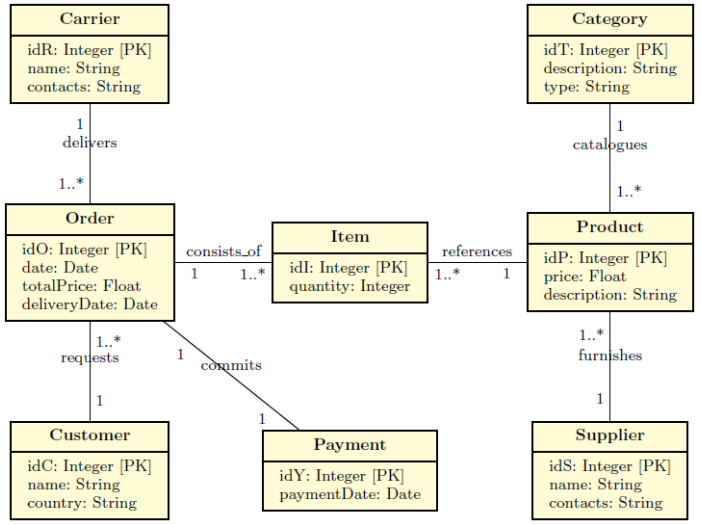




Exhaustive exploration

2) Generate collection root sets

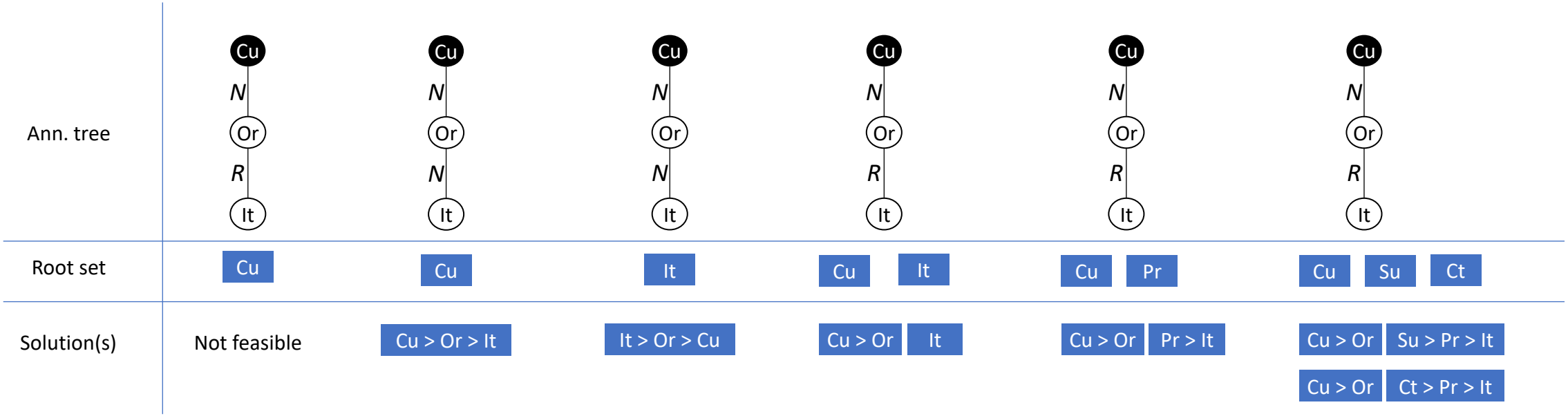
- A root set is a set of collections of which we know only the root entity
- Possible root sets with 3 entities
  - *Cu*
  - *Or*
  - *It*
  - *Cu, Or*
  - *Cu, It*
  - *Or, It*
  - *Cu, Or, It*



Exhaustive exploration

Given a root set *rs* and an annotated tree *t*, expand *rs* with nestings until *t* is satisfied

- *t* must be compatible with *rs* (e.g., can't have R edges if |*rs*|=*1*)



Exhaustive exploration

Finding all solutions

Algorithm 2 Exploration of document logical schemas

Require: RS: collection root sets; AT: annotated query trees
Ensure: S: solutions

1: $FS \leftarrow \emptyset$ ▷ Final solutions
2: $PS \leftarrow RS$ ▷ Partial solutions
3: **while** $PS \neq \emptyset$ **do**
4: $ps \leftarrow ps.pop()$
5: $i \leftarrow coalesce(ps.nextQuery, 0)$
6: **for all** $at \in AT[i] : at$ is compatible with ps **do**
7: $PS' \leftarrow$ expand ps to support at
8: **if** Iterated on all queries **then**
9: $FS \leftarrow FS \cup PS'$
10: **else**
11: $ps'.nextQuery \leftarrow ps'.nextQuery + 1 \ \forall ps' \in PS'$
12: $PS \leftarrow PS' \cup PS$
13: **return** FS

Finding only the best solution

Algorithm 3 Exploration of document logical schemas

Require: RS: collection root sets; AT: annotated query trees
Ensure: S: solutions

1: $bestSol \leftarrow \perp$ ▷ Best solution
2: $PS \leftarrow RS$ ▷ Partial solutions
3: **while** $PS \neq \emptyset$ **do**
4: $ps \leftarrow ps.pop()$
5: $i \leftarrow coalesce(ps.nextQuery, 0)$
6: **for all** $at \in AT[i] : at$ is compatible with ps **do**
7: $PS' \leftarrow$ expand ps to support at
8: **for all** $ps' \in PS : cost(ps') < cost(bestSol)$ **do**
9: **if** Iterated on all queries **then**
10: $bestSol \leftarrow ps'$
11: **else**
12: $ps'.nextQuery \leftarrow ps'.nextQuery + 1$
13: $PS \leftarrow \{ps'\} \cup PS$
14: **return** $bestSol$

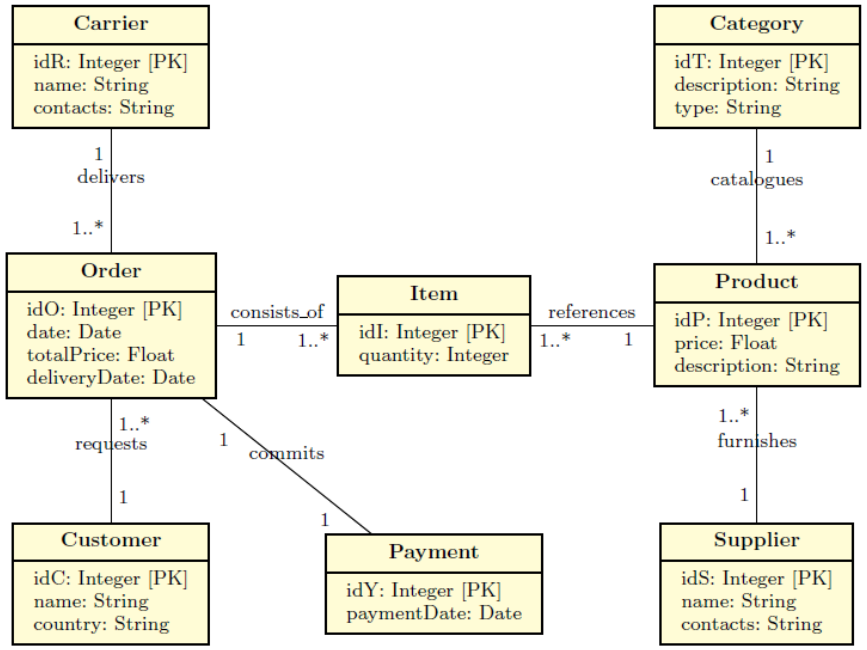
Exhaustive exploration

Results on OnlineStore (6 read queries)

|Q|	|AnnTrees|	|RootSet|	Exhaustive			Pruned		
			# explorations	# solutions	Time (s)	# explorations	# solutions	Time (s)
1	8	255	2.174	21	3	2.153	1	2
2	12	255	2.579	143	4	2.309	1	3
3	16	255	6.623	1.331	11	2.893	1	5
4	20	255	20.707	6.812	42	3.913	1	9
5	24	255	315.507	131.534	634	9.076	1	27
6	28	255	18.686.278	4.662.453	61.145	78.154	1	2.914

Heuristics

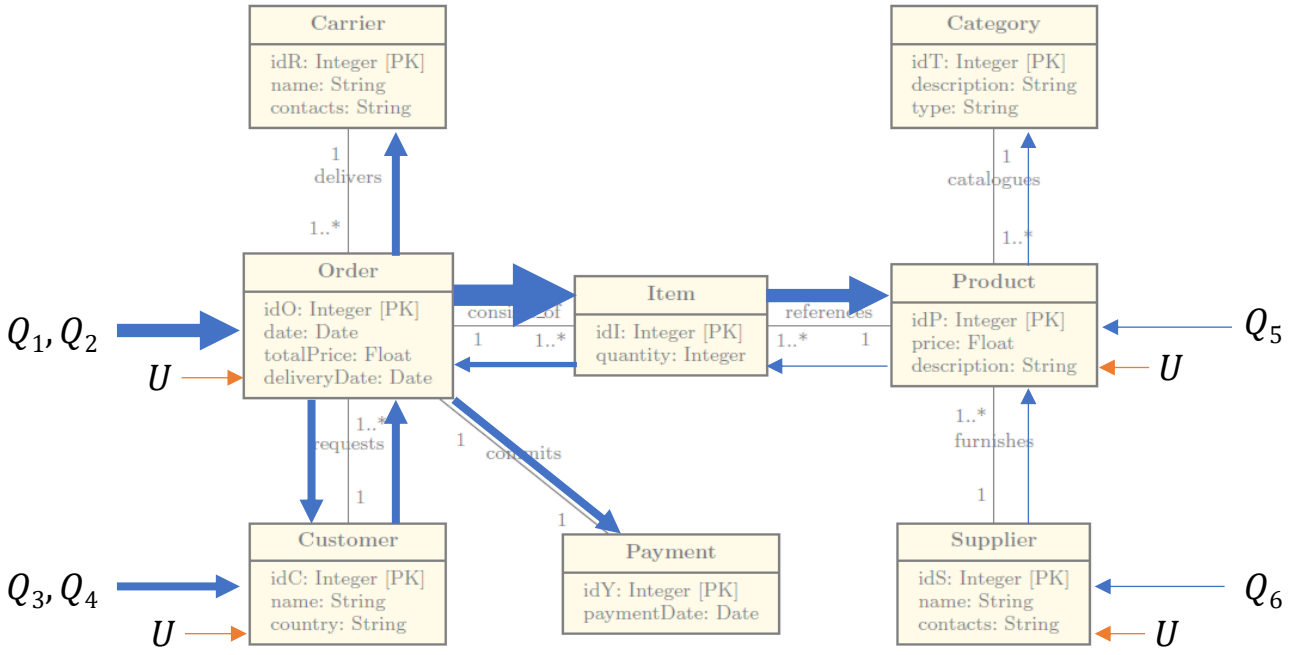
Now that we have the ground-truth, let's define a heuristics



Heuristics

Now that we have the ground-truth, let's define a heuristics

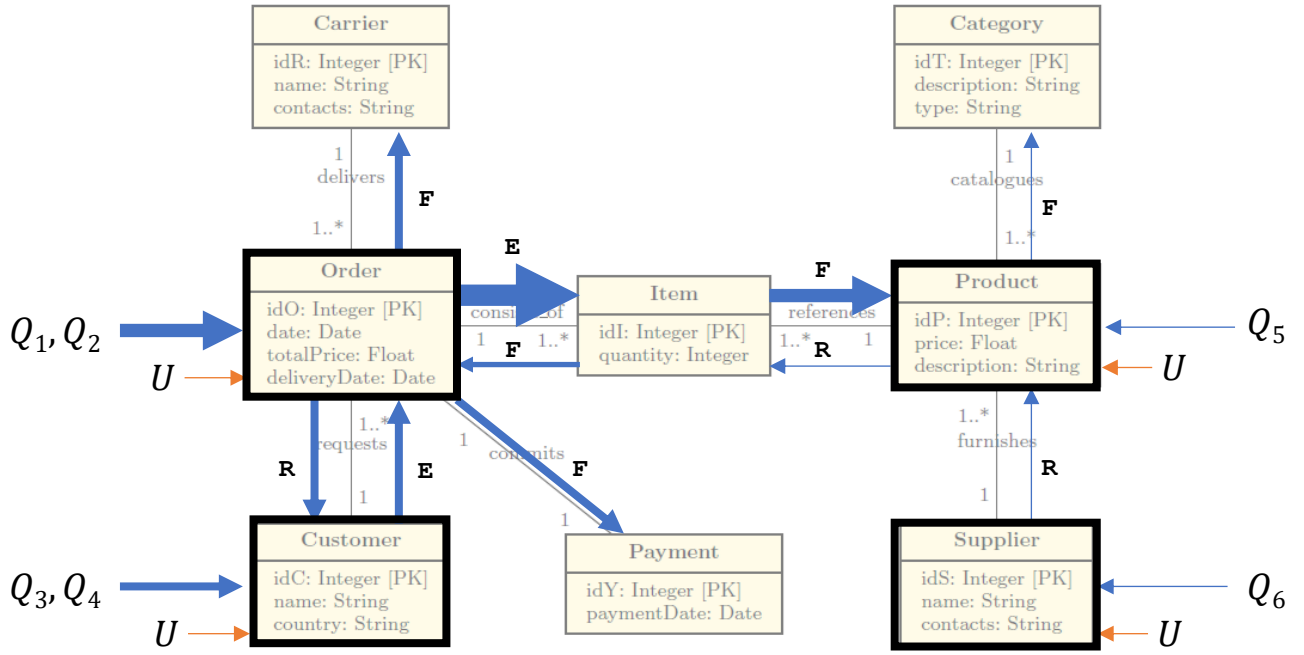
1. Quantify workload effort on entities and relationships



Heuristics

Now that we have the ground-truth, let's define a heuristics

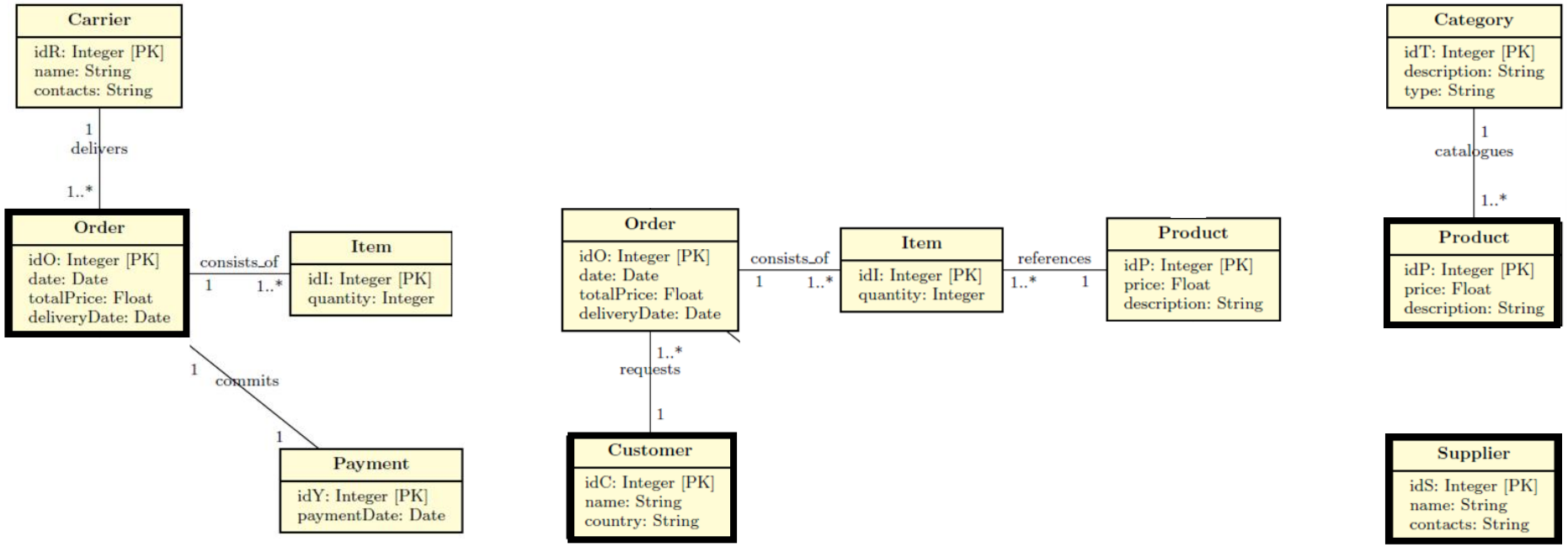
2. Use metrics to provide local recommendations



Heuristics

Now that we have the ground-truth, let's define a heuristics

3. Use recommendations to determine the logical schema
4. Validate against the (exhaustive?) exploration



Heuristics

Does it work…? ☺