POWER COMMUNITY

SLOW IS A DESIGN CHOICE





Microsoft Dynamics 365

AI ERP Summit

ENGINEERING THE FUTURE OF ERP WITH AGENTS

XPLUS GLOBAL · DOCENTRIC · ARQUICONSULT · DYNAMICS MINDS

COMMUNITY SPONSORS

FEATURED SPEAKER

Adrià Ariste Santacreu

Cloud Solution Architect, Microsoft

SESSION TITLE

Slow is a design choice

DATE
27th November

TRACK
D365 F&O

LOCATION
Lisbon, Portugal

Join me in Lisbon >

Learn. Connect. *Build The Future*



Performance problems are chosen

They're chosen, without anyone knowing they were choosing.

Production is just where they finally get noticed.

"It was fast at go-live"

- Anonymous



POWER COMMUNITY

The SQL was simple. The workload design was not.

OBSERVED



117 sec

Longest observed wait for a simple INSERT

What happened

Parallel posting sessions competed for the same database pages. The statement itself was simple; contention made it slow.

Response

STAGGER
posting threads

SPLIT
large journals

SCHEDULE
heavy runs off-peak

POWER COMMUNITY

29 seconds is a slow query. 1,226 executions is a design problem.

OBSERVED



29 sec
average duration

1,226
runs in one day

≈10 h
total database time

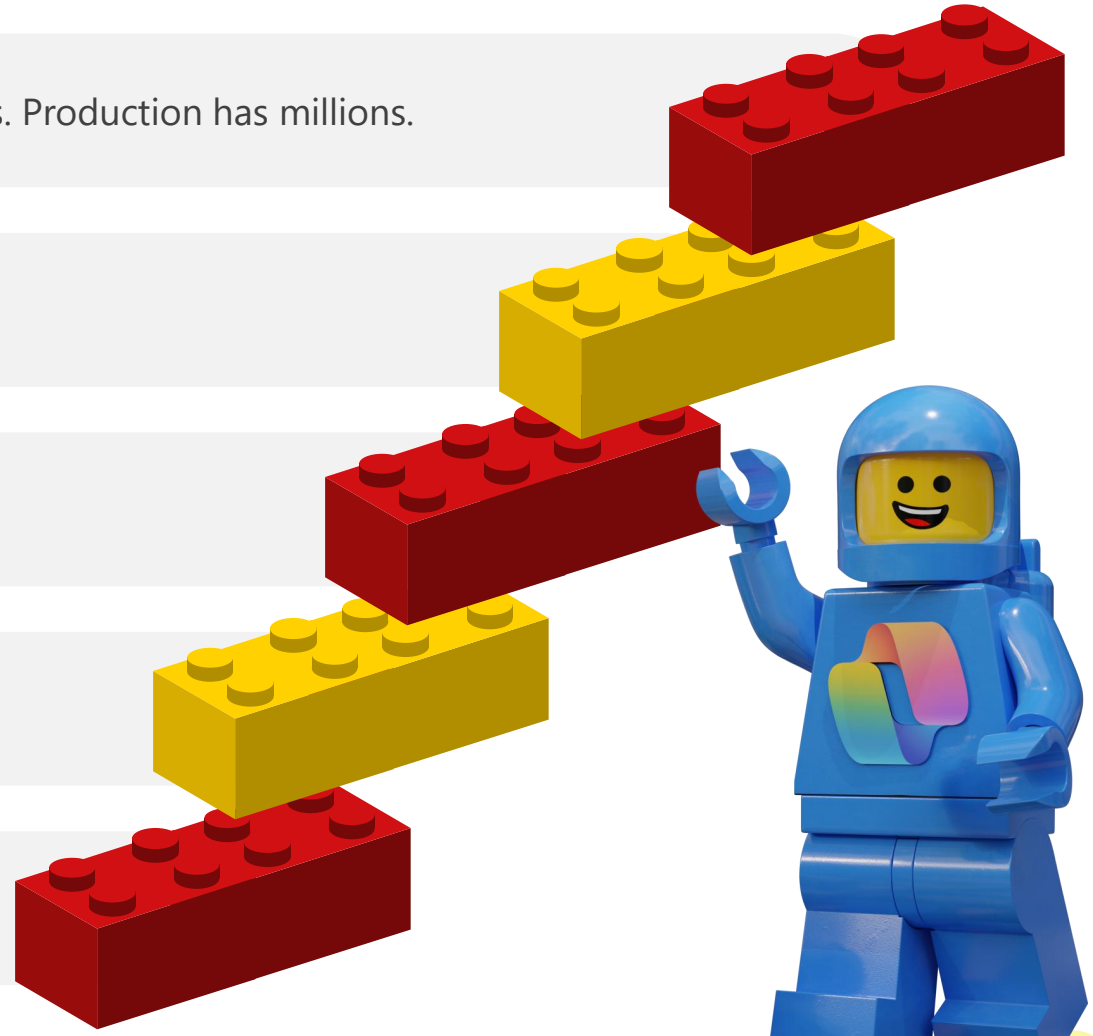
The query joined several large inventory tables and sorted by creation date.

Narrow candidates first · Remove unnecessary joins and sorting · Validate indexes

POWER COMMUNITY

Why tests don't catch it

Small data	Dev and test have a few thousand clean rows. Production has millions.
Tests check "works"	Tests prove it works, not what it costs.
One user	A single tester hides locking and contention.
No growth yet	Data hasn't had a year to accumulate.
Admin role	Testing as admin hides XDS and security cost.



What we'll walk through

Data model, cache and indexes

Tables

X++ code

Patterns that read well and scale badly

Display methods, XDS, personalization

Forms



Integrations

OData, entities and throughput

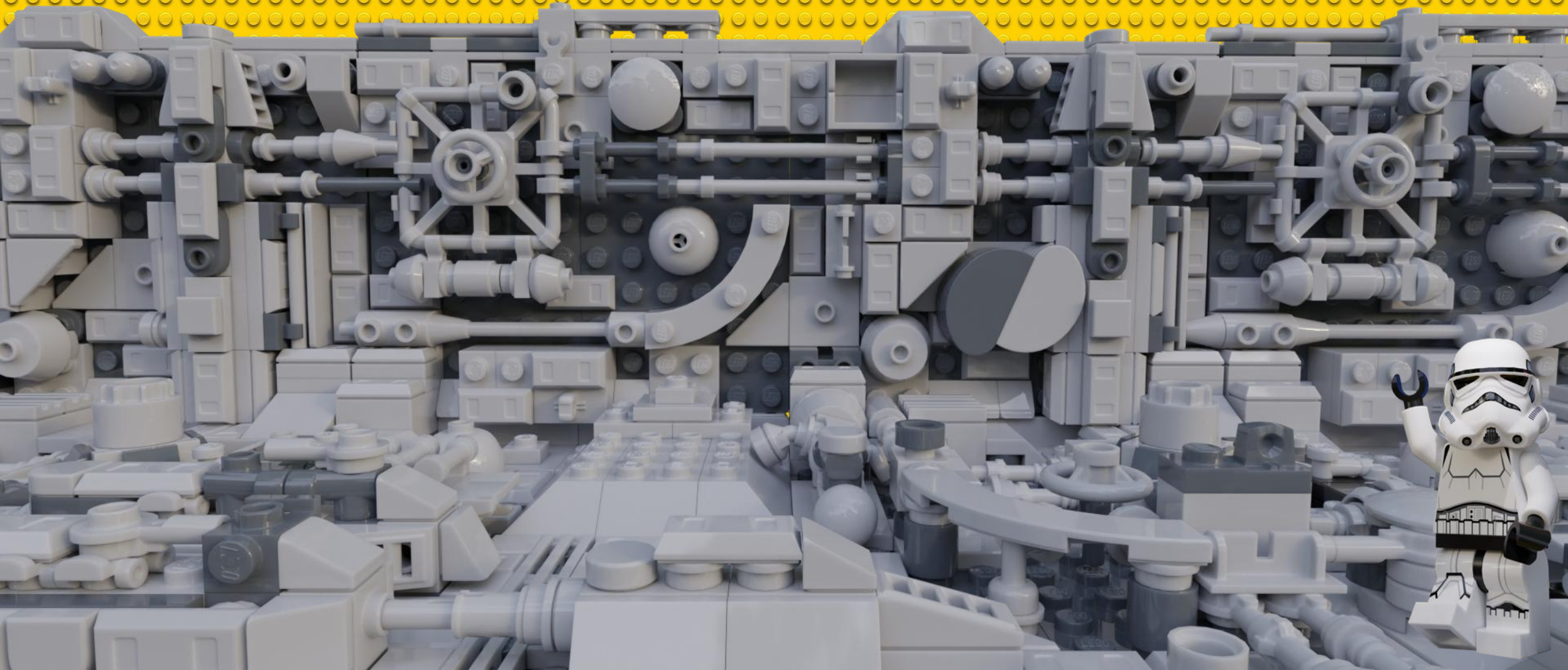
Settings that look like admin

Configuration

Other things

Extensions, batch, testing, AI

01 · Tables & data model



The index that passes every test



FOUND IN REVIEW

The issue

The index exists, but doesn't match how the table is really queried.

- Wrong field order
- Low-selectivity leading fields

How it shows up

Fine with 5,000 rows. With 30 million: scans and high logical reads.

- Tops Query Store
- Only in production

Fix / prevent

Design indexes from real query shapes, not from the table.

- Equality first, ranges later
- Test real volume and distribution

Cache settings chosen by habit



FOUND IN REVIEW

The issue

CacheLookup left at None on hot setup tables, or EntireTable on big, volatile ones.

- Default never reviewed
- "Caching is always good"

How it shows up

Thousands of identical tiny queries in traces.

- Same SELECT x10,000
- Unexpected load

Fix / prevent

Choose from volatility and access pattern.

- Found / FoundAndEmpty for stable key lookups
- EntireTable only for small, static tables

Tables with no exit plan



FOUND IN REVIEW

The issue

Custom log, history or staging tables created with no cleanup or archive design.

- "We'll clean it later"
- No owner

How it shows up

Years later it's one of the biggest tables in the database.

- Slower queries and maintenance
- Capacity costs

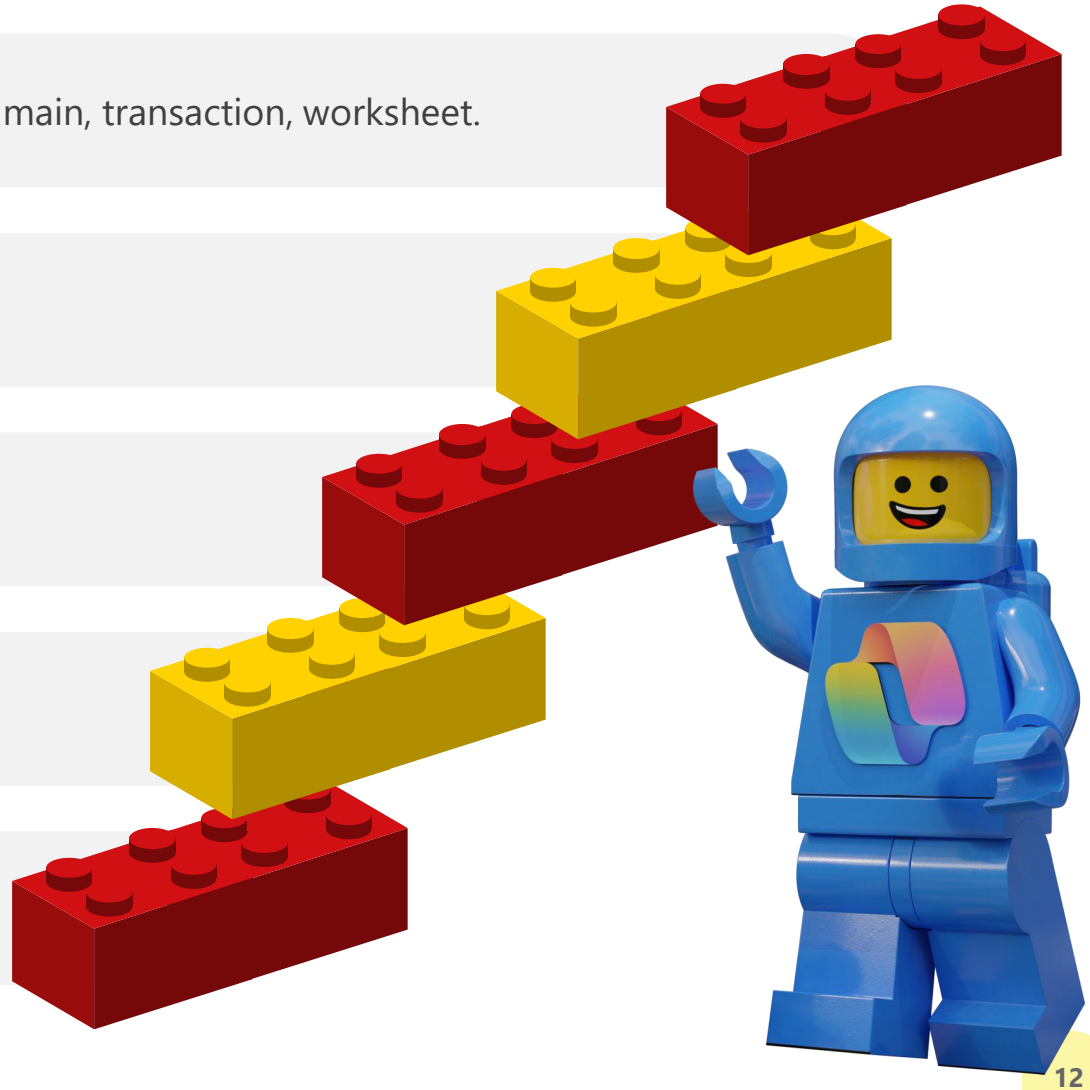
Fix / prevent

Define retention when the table is created, not when storage alerts fire.

- Cleanup job from day one
- Clear data owner

Also watch for

Table group	Doesn't reflect what the table really is: setup, main, transaction, worksheet.
Too many indexes	Every insert and update pays for each one.
Duplicate indexes	Overlapping custom indexes nobody uses.
Wide tables	Rarely used fields read on every select.
Missing relations	Wrong joins and poor plans.

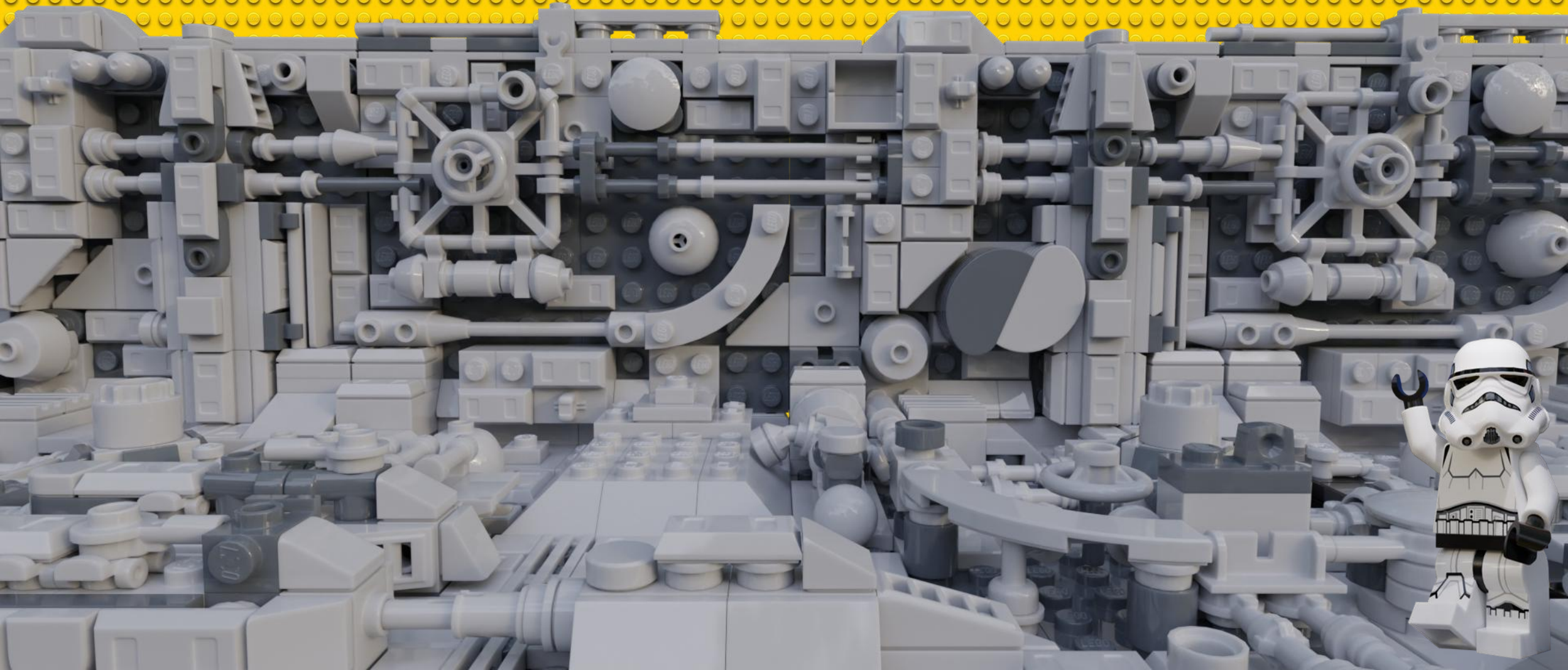




Takeaway

Metadata is runtime behavior. The table properties you click through in Visual Studio decide how SQL works in production.

02 · Inefficient X++



Loop + if



FOUND IN REVIEW

The issue

Filtering happens in X++ instead of SQL.

- Rows fetched only to be discarded

How it shows up

More rows transferred, more round trips, more AOS memory.

- Each fetch looks fast in a trace

Fix / prevent

If the condition fits in the where clause, put it there.

- Let SQL do the filtering

POWER COMMUNITY

Loop + if: let SQL filter

POWER COMMUNITY

PROBLEM

```
// SQL returns everything,
// X++ throws most of it away
while select salesLine
    where salesLine.SalesId == _salesId
{
    if (salesLine.SalesStatus
            == SalesStatus::Backorder
        && salesLine.RemainSalesPhysical > 0)
    {
        this.process(salesLine);
    }
}
```

BETTER

```
// Filter in the where clause
while select salesLine
    where salesLine.SalesId == _salesId
       && salesLine.SalesStatus
            == SalesStatus::Backorder
       && salesLine.RemainSalesPhysical > 0
{
    this.process(salesLine);
}
```



Nested loops and select inside a loop



FOUND IN REVIEW

The issue

Code reads naturally, but database calls grow with the data.

- find() per row
- Inner while select per row

How it shows up

2 minutes at go-live, 2 hours a year later.

- Same statement thousands of times in the trace

Fix / prevent

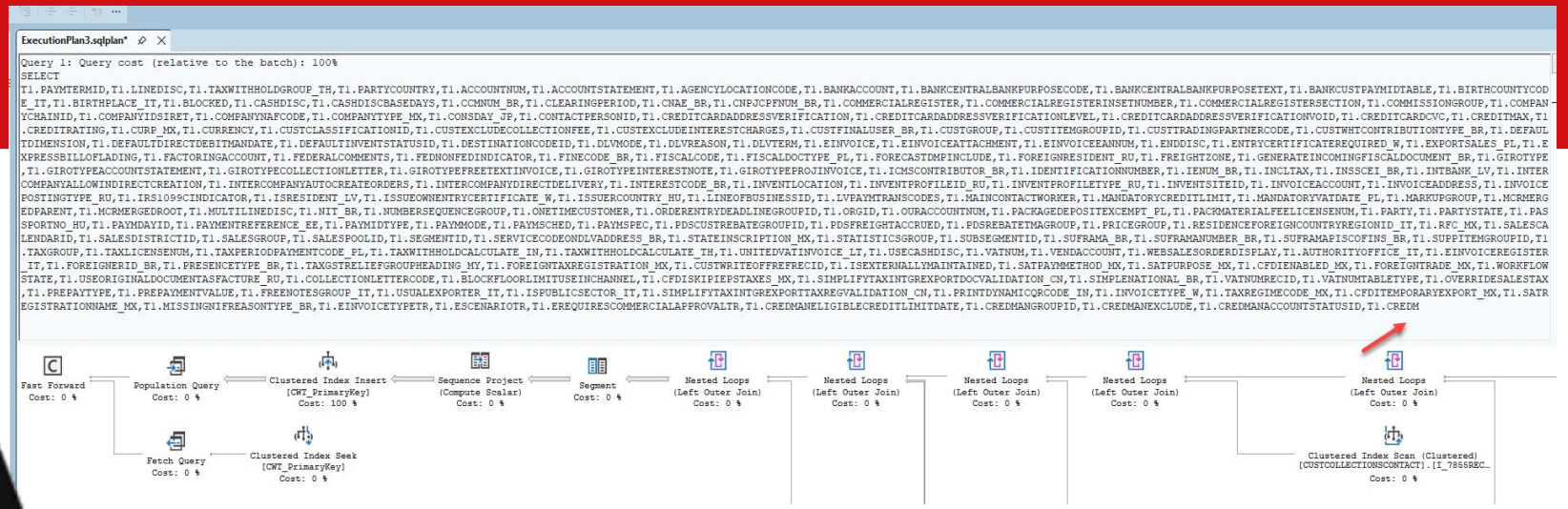
Ask: how many queries for 100,000 rows?

- Joins, exists / notexists join
- Field lists, cache lookups in a map

OBSERVED

Four loops. One exploding query.





Nested loops: one join

POWER COMMUNITY

PROBLEM

```
// 1 query + 1 per order + 1 per customer
while select salesTable
    where salesTable.SalesStatus
        == SalesStatus::Backorder
{
    custTable = CustTable::find(
        salesTable.CustAccount);

    while select salesLine
        where salesLine.SalesId
            == salesTable.SalesId
    {
        // ...
    }
}
```

BETTER

```
// One query, only the fields needed
while select SalesId, CustAccount
    from salesTable
    where salesTable.SalesStatus
        == SalesStatus::Backorder
join CustGroup from custTable
    where custTable.AccountNum
        == salesTable.CustAccount
join ItemId, SalesQty from salesLine
    where salesLine.SalesId
        == salesTable.SalesId
{
    // ...
}
```



Set-based, until it isn't



FOUND IN REVIEW

The issue

Row-by-row where update_recordset would work, or set-based code that is downgraded to row-by-row.

- Overridden insert / update / delete
- Database log, alerts

How it shows up

The code looks set-based, but runs row by row.

- Same duration as a loop

Fix / prevent

Use set-based operations and check for fallback causes.

- skipDataMethods / skipDatabaseLog only when logic allows it
- Document why

POWER COMMUNITY

Select what needs to be selected



The issue

SELECT * FROM TABLE;

"It's only a few columns"

How it shows up

A select with 498 selected columns across different joins.

FOUND IN REVIEW

Fix / prevent

Select only the fields you need.

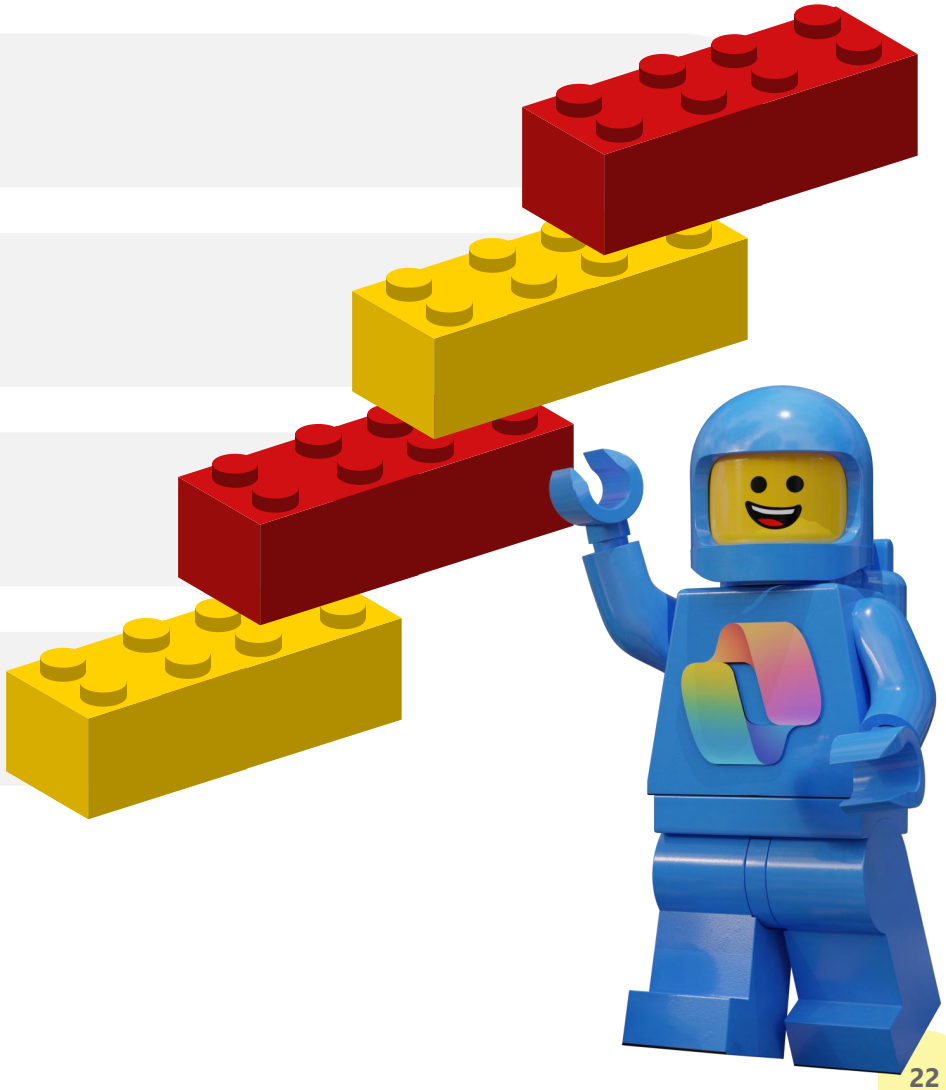
- 10 ms × 1 = 10 ms

10 ms × 1,000,000

= 10,000 sec ≈ 2 h 47 min

Also watch for

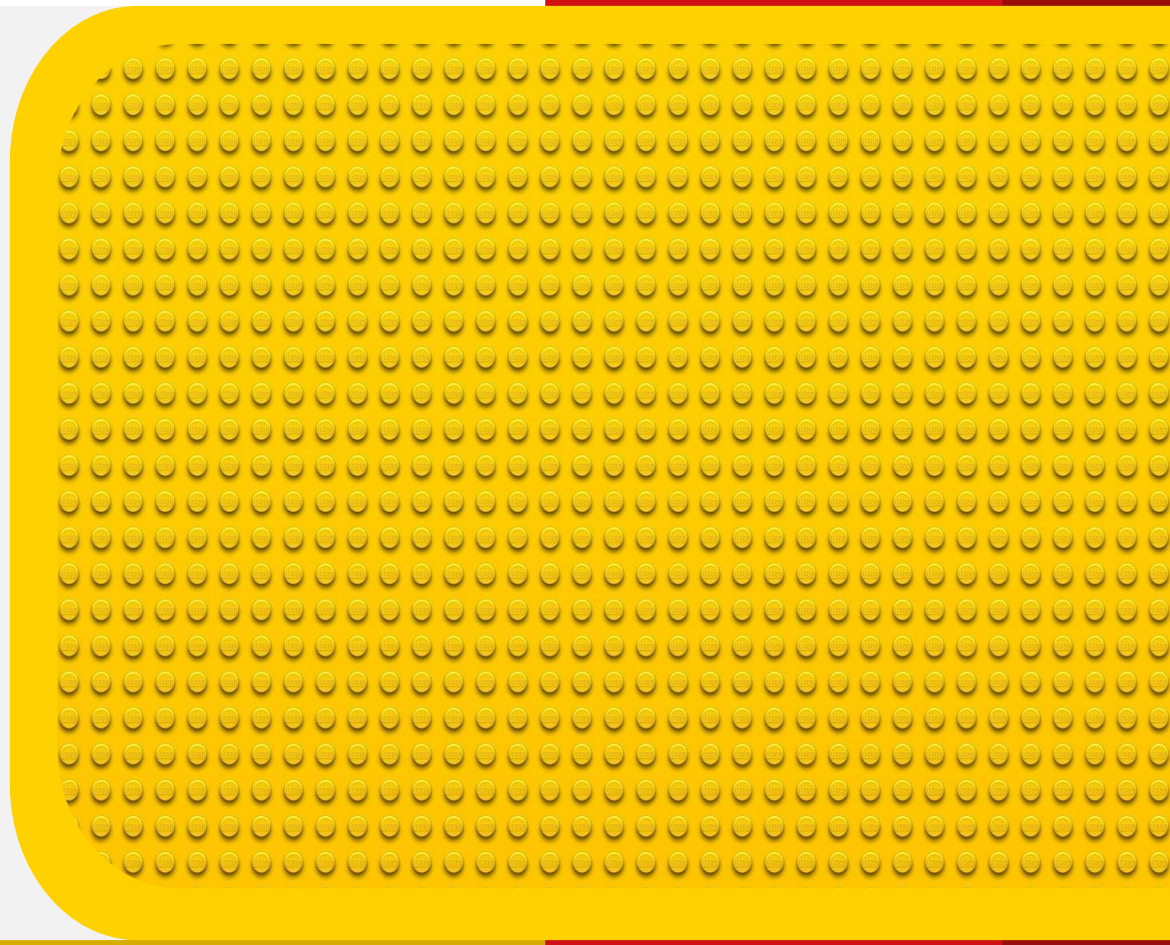
Select *	Field lists when you need two fields.
find() in loops	Repeated single-record lookups.
crossCompany	Without limiting the companies.
Huge tts scopes	Locks held for minutes.





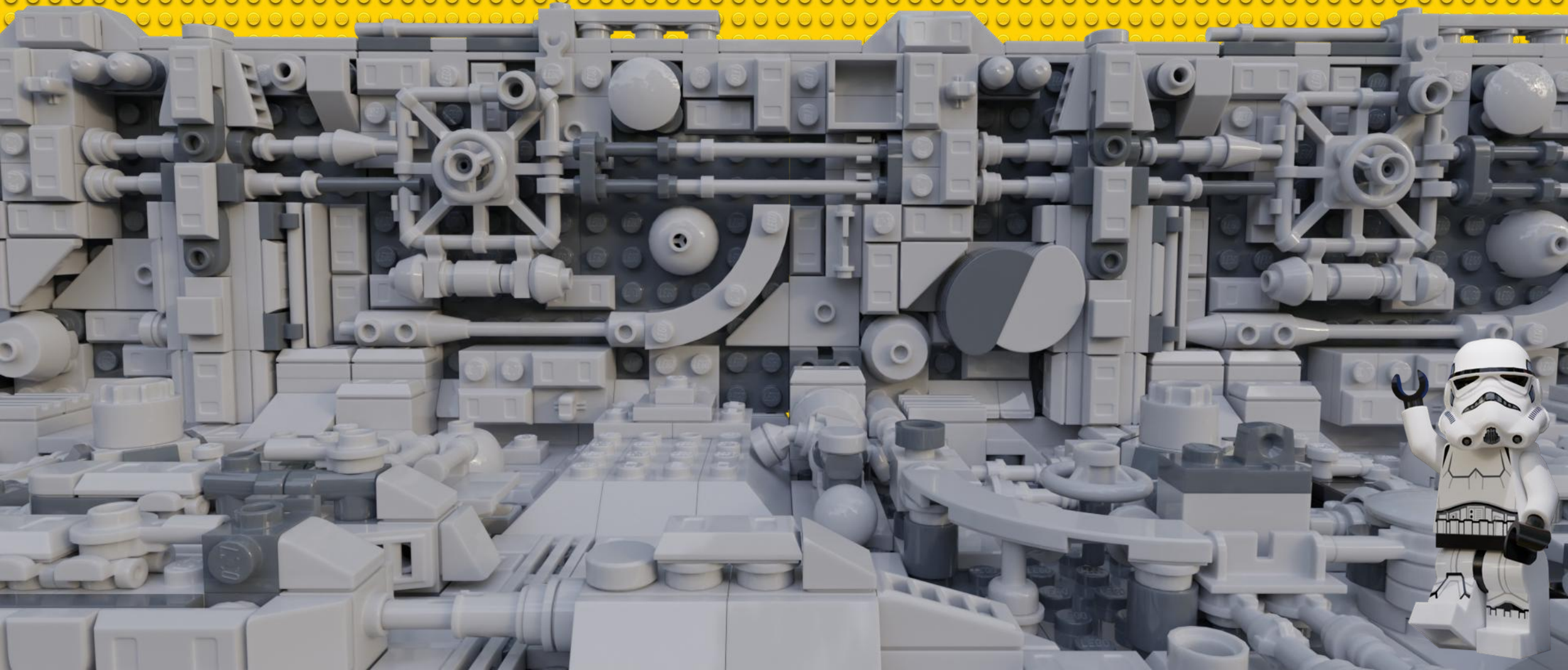
Takeaway

Code that reads well isn't necessarily code that scales. Always ask: how many times does this hit the database?



POWER COMMUNITY

03 · Forms



Display methods: the cost is per row



FOUND IN REVIEW

The issue

A display method runs a query for every visible row, again on scroll and refresh.

- Looks like one small method

How it shows up

One select x 50 rows x refreshes = hundreds of queries to open a grid.

- Slow grids, slow scrolling

Fix / prevent

Join the data in the datasource instead.

- cacheAddMethod where appropriate
- Views or computed columns

POWER COMMUNITY

XDS policies

DESIGN GUIDANCE

The issue

The policy query is added to every query on constrained tables: forms, lookups, reports, code.

- Security designed without load in mind

How it shows up

Fast for admin, slow for real users.

- Complex execution plans
- Only reproduces with specific roles

Fix / prevent

Security design is performance design.

- Simple, indexed policy queries
- Always test with real roles





Excessive personalization



Microsoft Dynamics 365 Unified Operations: Trace Parser - released_product2__2026_09_18_07_07_51 (1)

File Edit View Help Timeline View Editor

Select Grouping [OneBox](143@2980@@{5BEA514B-F4D8-44F3-9A04-193B6EEAA005

OBSERVED

Timeline Overview Call Tree X++ SQL

Color by Inclusive

Call Stack	Inclusive (ms)	Exclusive (ms)	RPC Calls	Database Calls	Database Time (ms)
Command::::::	0.01	0.01	0	0	0.00
Command::::::	0.00	0.00	0	0	0.00
Command::::::	0.01	0.01	0	0	0.00
Command::::::	0.00	0.00	0	0	0.00
Command::::::	0.01	0.01	0	0	0.00
Command::::::	0.01	0.01	0	0	0.00
Command::::::	0.01	0.01	0	0	0.00
Command::::::	0.01	0.01	0	0	0.00
Command::::::	0.01	0.01	0	0	0.00
Command::::::	26,967.77	0.00	0	28	25,853.55
PropertyChange::::::	26,967.77	0.00	0	28	25,853.55
PropertyChange::::::	26,967.77	0.00	0	28	25,853.55
PropertyChange::::::	26,967.77	0.00	0	28	25,853.55
PropertyChange::::::	26,967.77	0.00	0	28	25,853.55
PropertyChange::::::	26,967.77	0.00	0	28	25,853.55
PropertyChange::::::	26,967.77	0.00	0	28	25,853.55
PropertyChange::::::	26,967.77	0.00	0	28	25,853.55
PropertyChange::::::	26,967.77	0.00	0	28	25,853.55
PropertyChange::::::	26,967.77	0.00	0	28	25,853.55
PropertyChange::::::	26,967.77	0.00	0	28	25,853.55
Info::startup	20.87	0.02	0	3	8.17



Excessive personalization

OBSERVED

The issue

Users add columns, display methods, and joins with filters on non-indexed fields.

- No governance of shared views

How it shows up

"It's slow for me but not for my colleague."

- Same form, same code, different workload

Fix / prevent

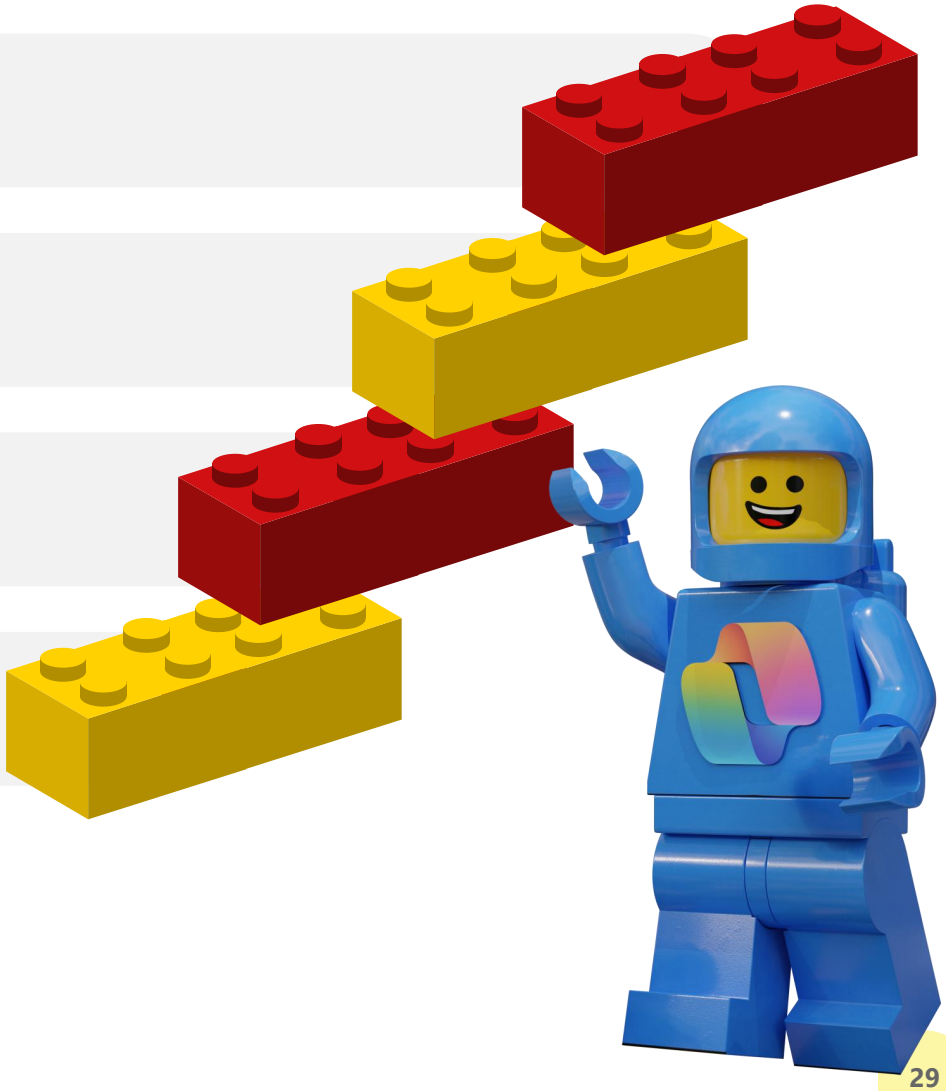
Reproduce with the user's view first.

- Then compare with a clean one
- Fast default views, key-user education



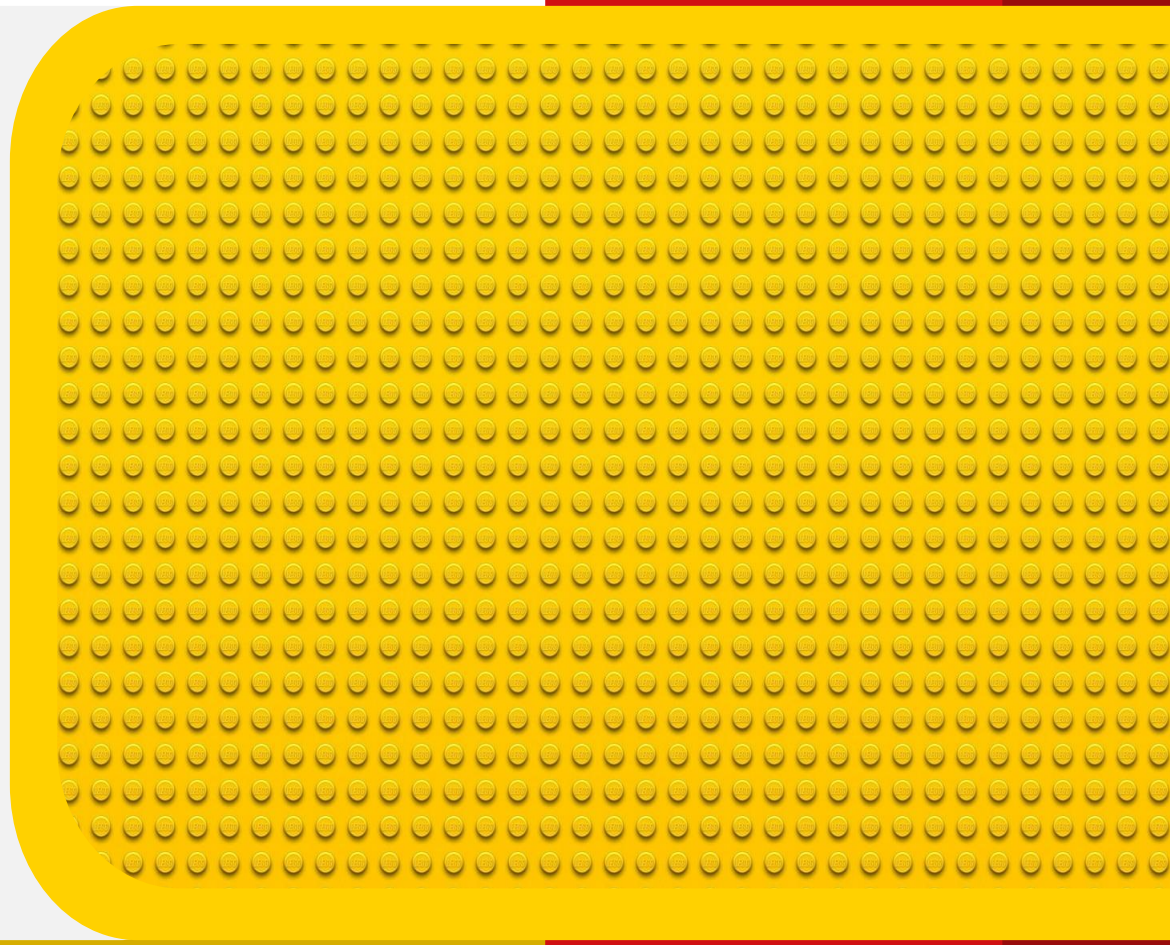
Also watch for

Datasource joins	Unneeded joins, outer joins where inner would do.
research()	Repeated executeQuery() / research() on events.
Lookups	Returning the whole table.
active()	Heavy logic on every row change.

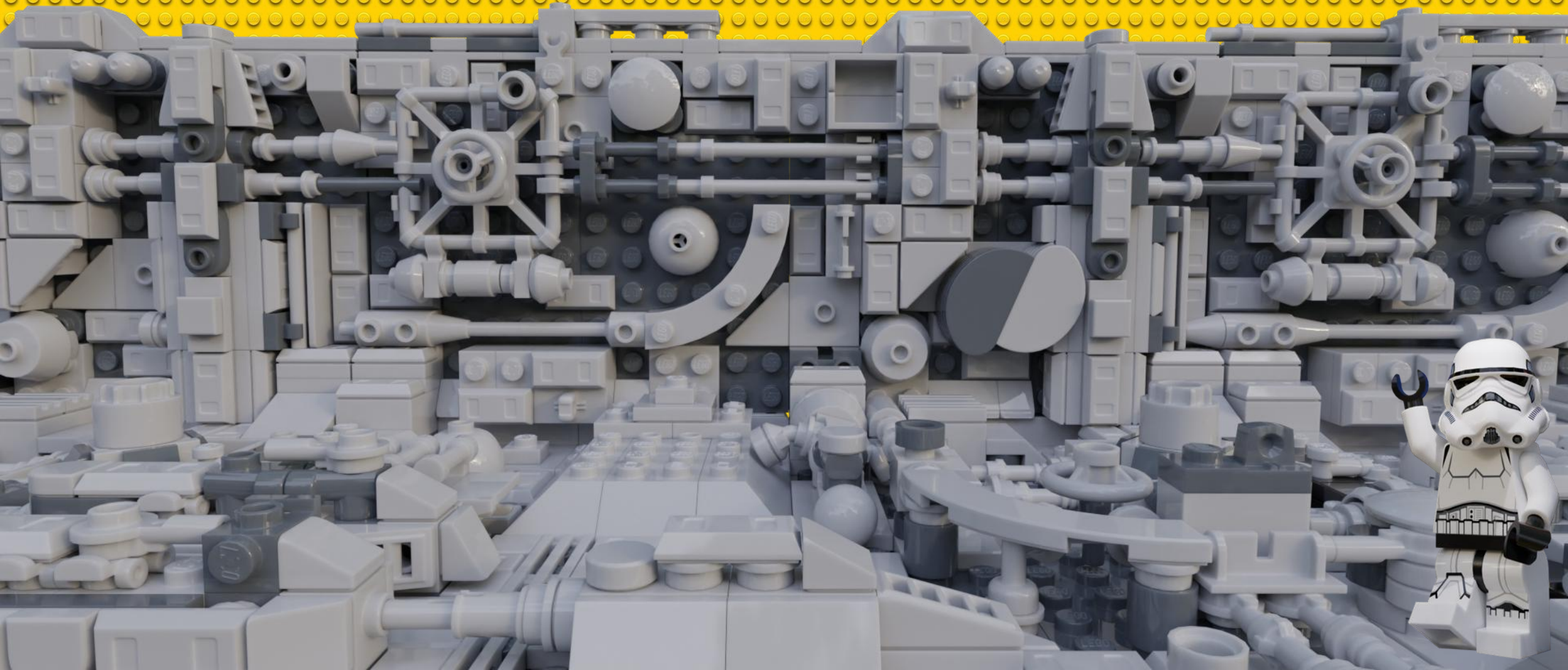


Takeaway

A form is not one query. It's every row, every column, every refresh, and every user's personalization.



04 · Integrations & entities



OData for everything



DESIGN GUIDANCE

The issue

OData used for high-volume or bulk integrations because it was quick to build.

- Request per record

How it shows up

Throttling and long-running integrations.

- Competes with interactive users

Fix / prevent

Choose the pattern from volume and timing.

- DMF / package APIs for bulk
- Business events + pull for notifications

POWER COMMUNITY

The convenient entity



FOUND IN REVIEW

The issue

Many datasources, nested views, virtual fields, logic in postLoad().

- Built for the first use case

How it shows up

100 records: fine. 2 million: hours.

- Row-by-row processing, no set-based

Fix / prevent

Purpose-specific entities.

- Minimal datasources and fields
- Computed columns over virtual fields

POWER COMMUNITY

The convenient entity



DESIGN GUIDANCE

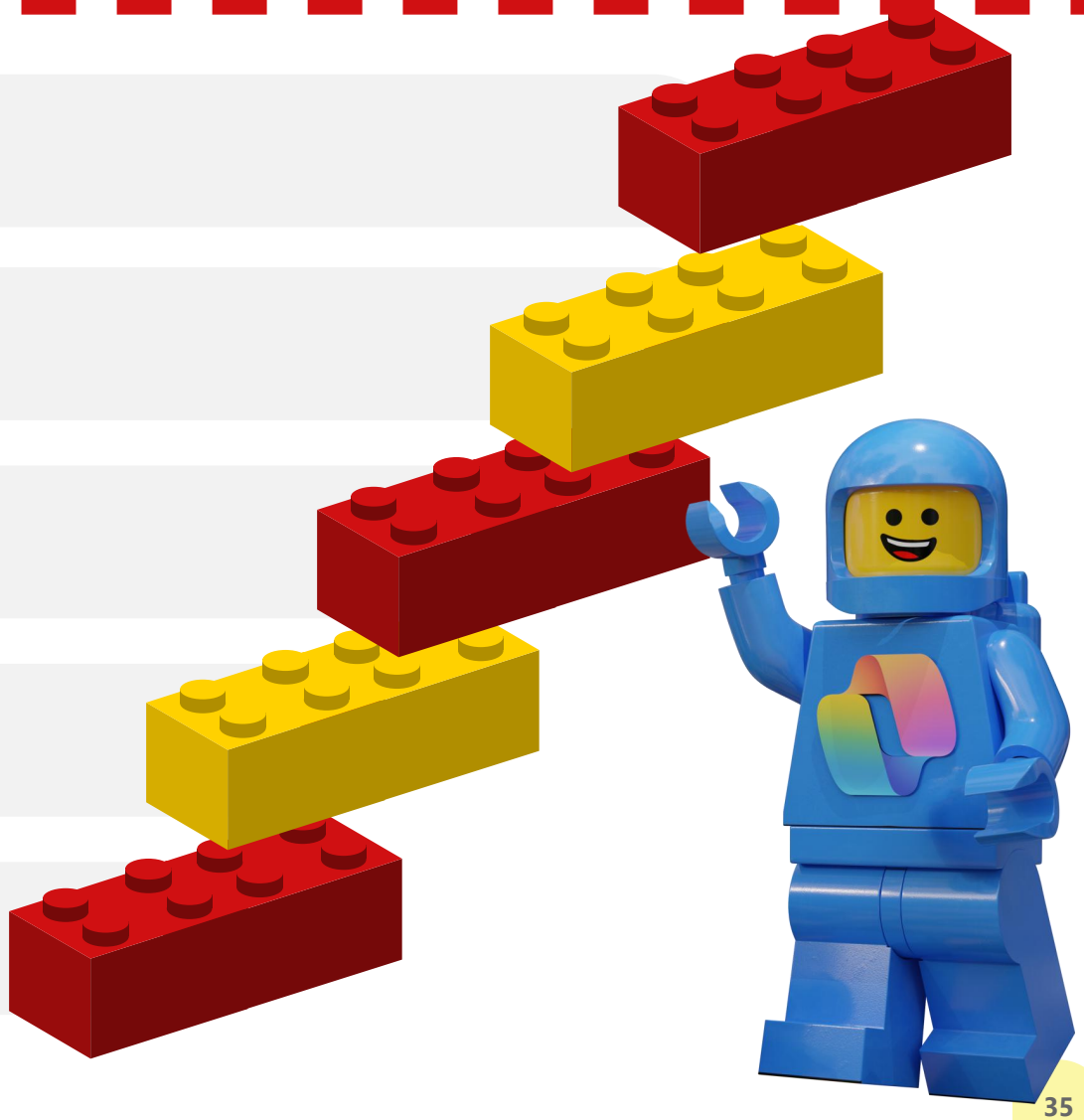
Help

Nathan Clouse's "The Odata Experiment":

https://www.atomicax.com/content/odata-experiment-deep-dive-odata-and-how-use-it

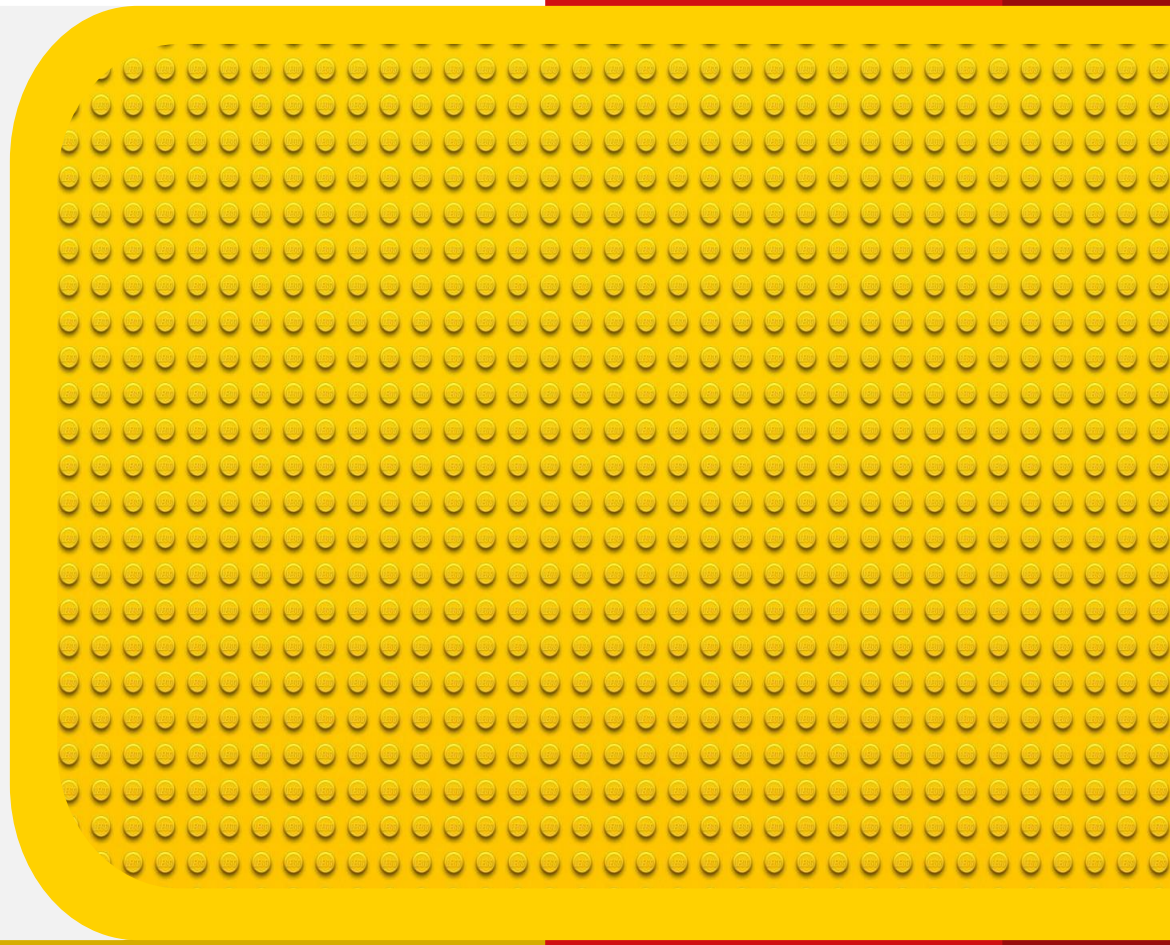
Also watch for

Polling	Avoid polling OData. Use publish/subscribe.
Every minute	Integrations scheduled "just in case".
Concurrency	Everything runs at the same time.
Staging	Staging tables never cleaned.
Custom services	Same X++ anti-patterns.



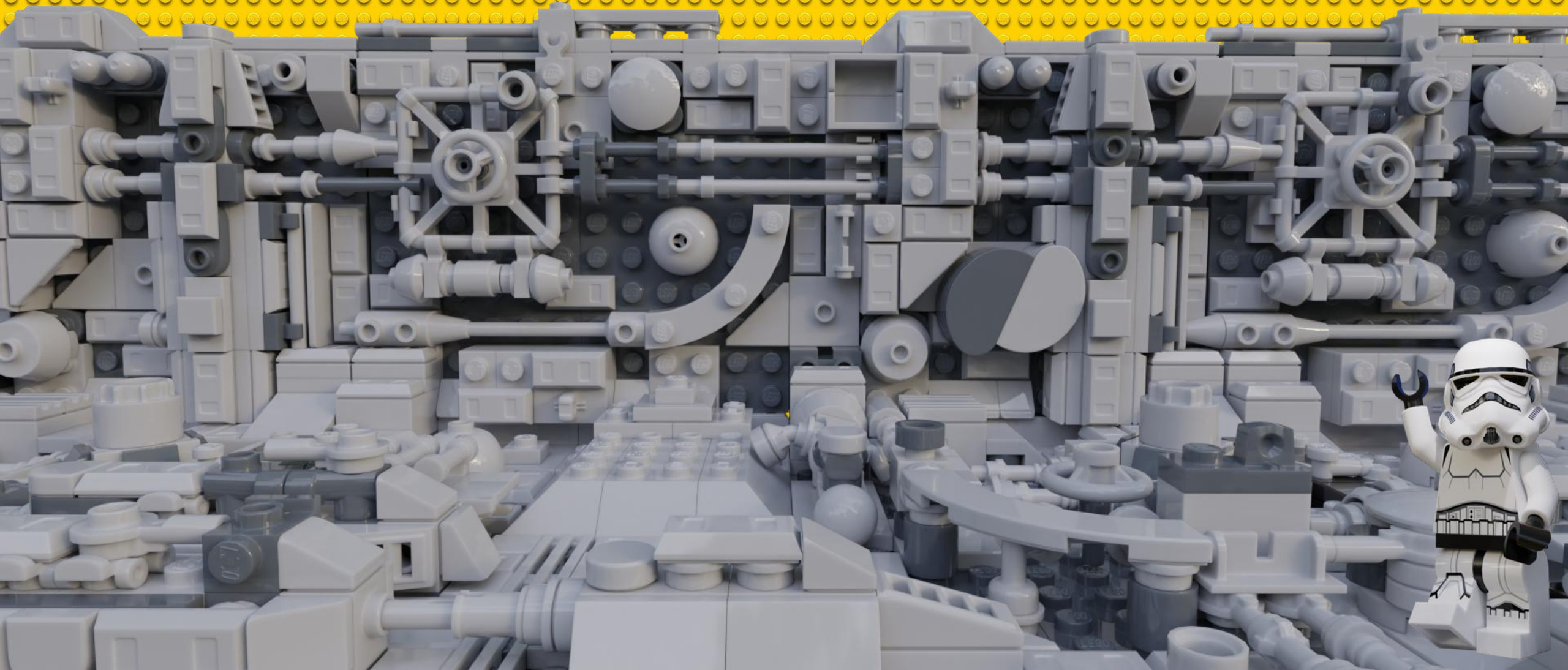
Takeaway

An entity isn't just a contract. It's code that runs for every record you send through it.



POWER COMMUNITY

05 · Configuration



OBSERVED

Database logging



The issue

Logging enabled broadly "for audit", including high-volume transaction tables.

- No owner, no justification

How it shows up

Extra writes, growing SysDatabaseLog, set-based ops falling back to row by row.

- Slower with no code change

Fix / prevent

Log only what's required.

- Avoid high-volume tables
- Clean up, review before go-live

Continuous number sequences



DESIGN GUIDANCE

The issue

Continuous numbering used without a documented legal or business requirement.

How it shows up

More database interaction and contention at high volume.

- Historically worst under concurrency

Fix / prevent

If gapless is not required, use non-continuous. If it is, use modern prefetching and test expected concurrency.

- Performance feature on by default from version 10.0.36

OBSERVED

Batch configuration



The issue

Everything in the default group, heavy jobs overlapping, more threads = faster.

- No workload design

How it shows up

Critical jobs start late and block each other.

- Users slow in batch windows

Fix / prevent

Design batch groups and priorities.

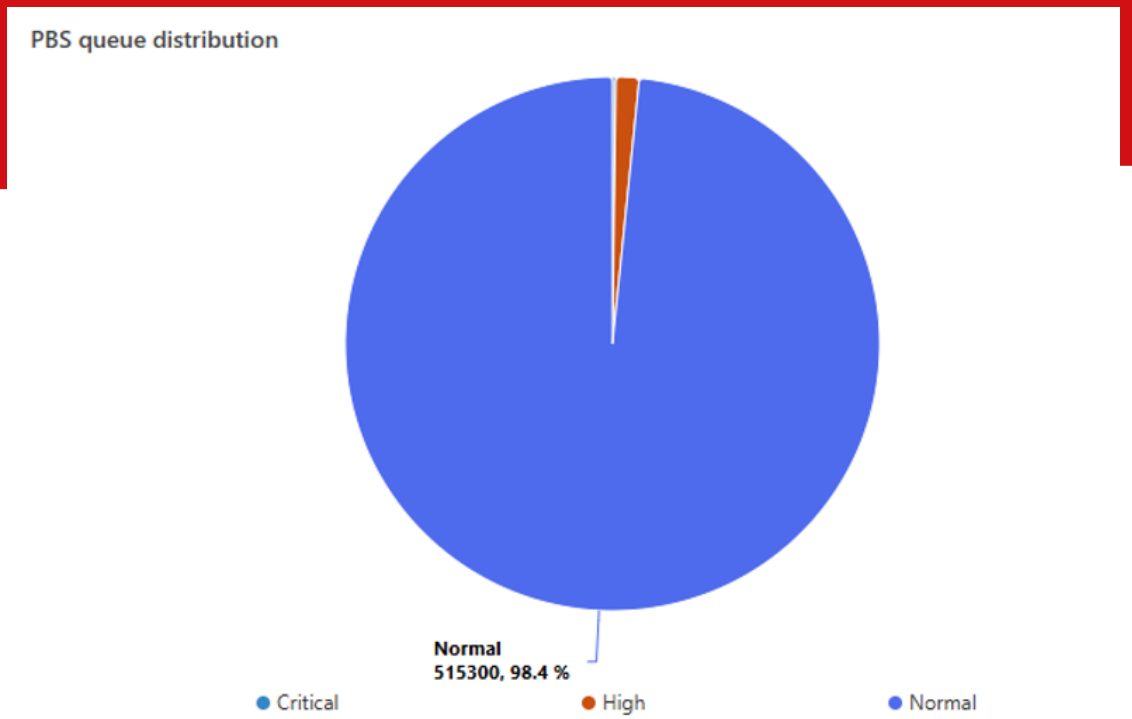
- Schedule by workload
- Tune parallelism to the process

OBSERVED

98.4% in the default queue

515,300 tasks marked Normal





Default priority is not a workload strategy.

OBSERVED

Batch configuration



The issue

Multiple instances of the same batch running at the same time.

- Record locking

How it shows up

Batch jobs never end, or take longer.

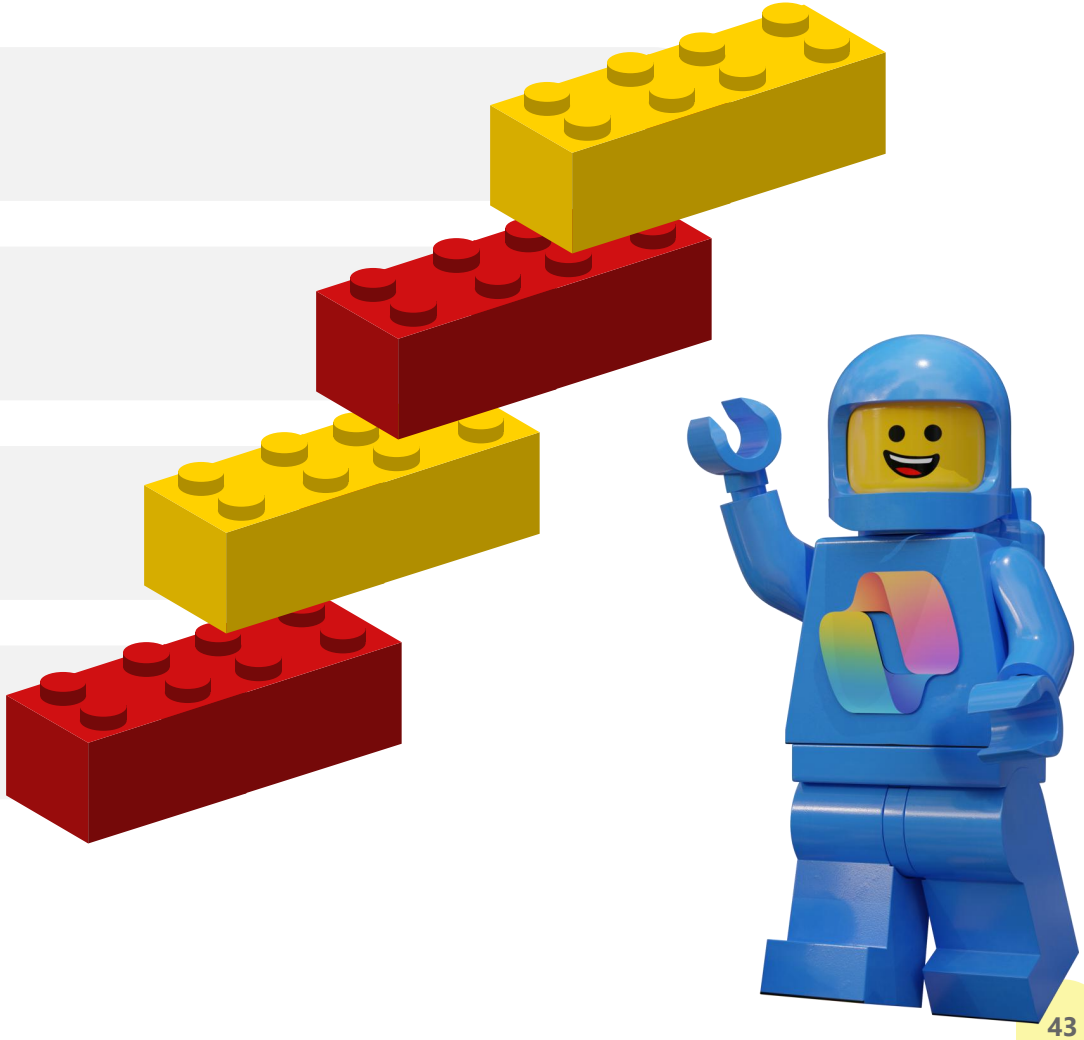
Excessive parallelism makes runs slower instead of faster!

Fix / prevent

- Use “Max Concurrency” in batch groups.
- If needed, tune parallelism to the process

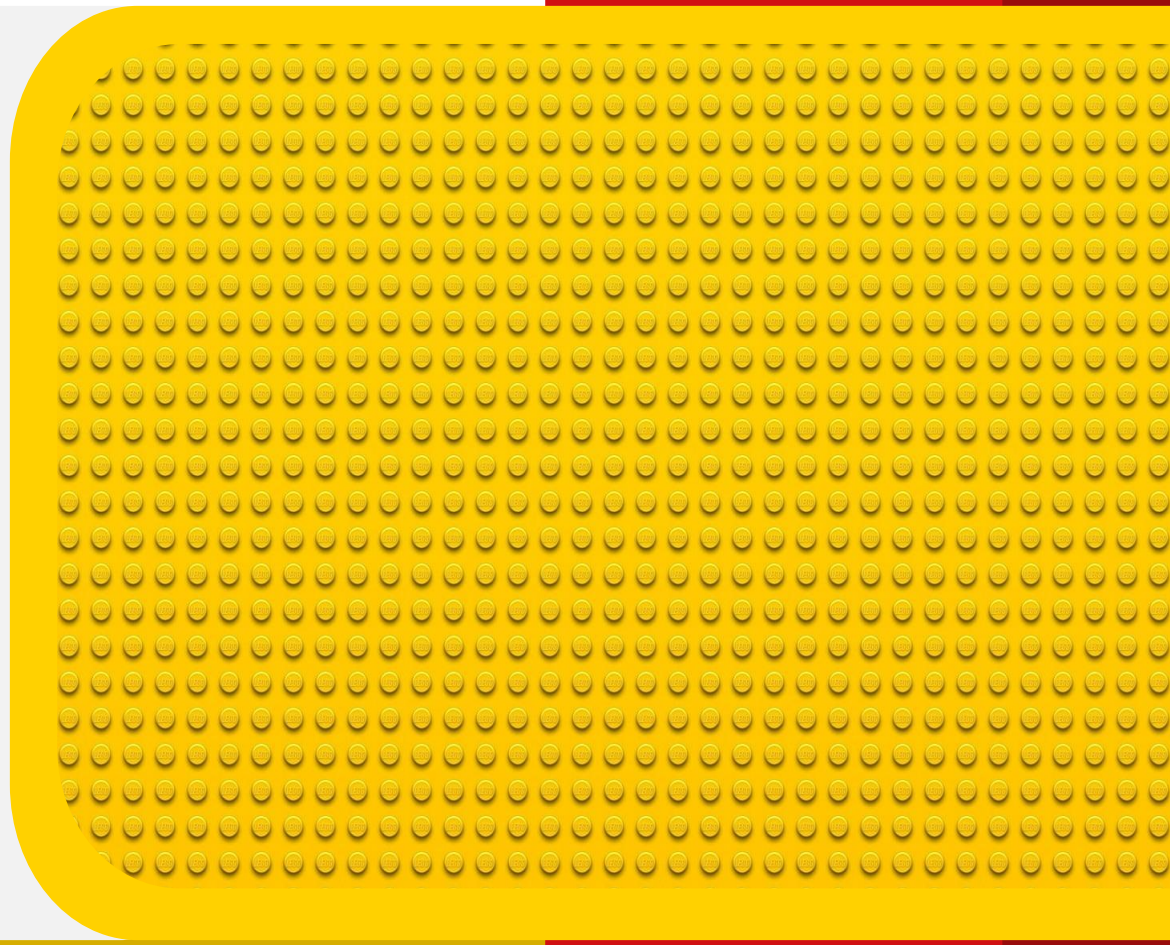
Also watch for

Cleanup jobs	Standard cleanups never scheduled.
Retention	Not defined, so data just grows.
Features	Enabled without knowing their impact.
Tracking	Logging nobody reads.



Takeaway

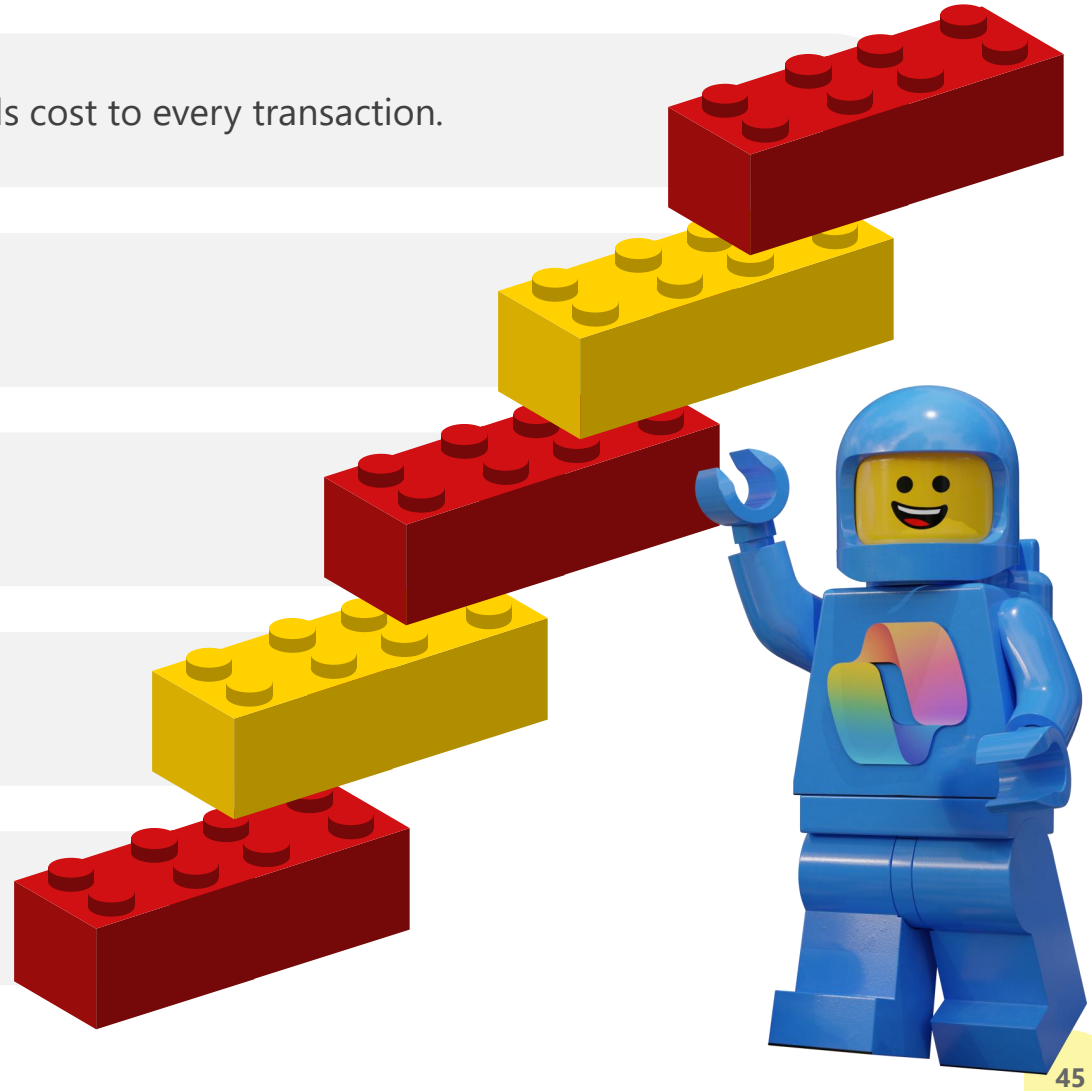
If a setting changes how F&O reads, writes, locks or schedules work, it's not admin. It's design.



POWER COMMUNITY

Other things that bite

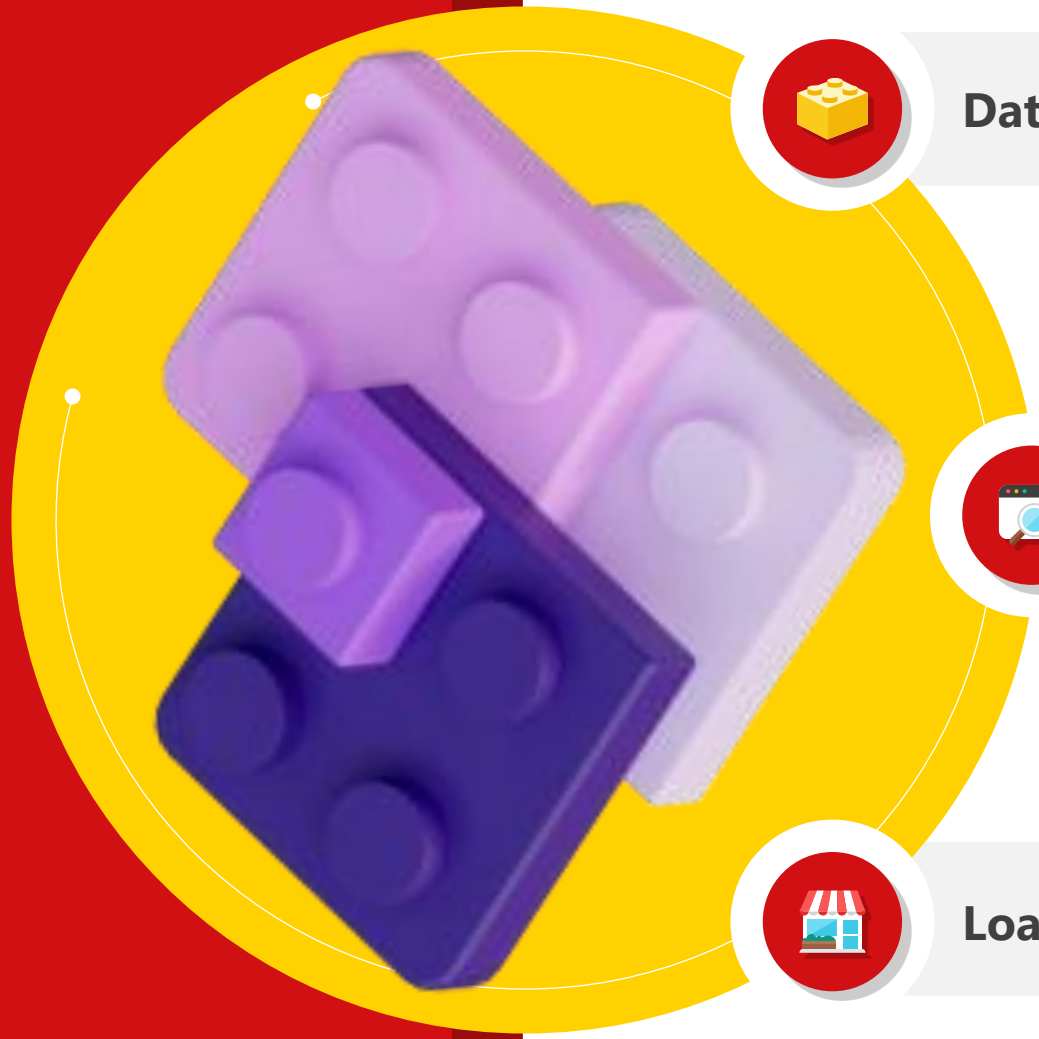
CoC on hot methods	Extending validateWrite, insert or pricing adds cost to every transaction.
Event handlers	"Small" logic on very frequent events.
Reporting	Heavy queries on operational tables.
Batch tasks	Poor partitioning: one task does 90%.
Perf testing	Fake data, one user, or none.



AI won't save you 😜

AI can spot loop + if, suggest joins, explain execution plans and speed up code reviews.

But AI helps you write code faster. It doesn't make the code run faster.



Data

Doesn't know your volumes or distribution

Roles

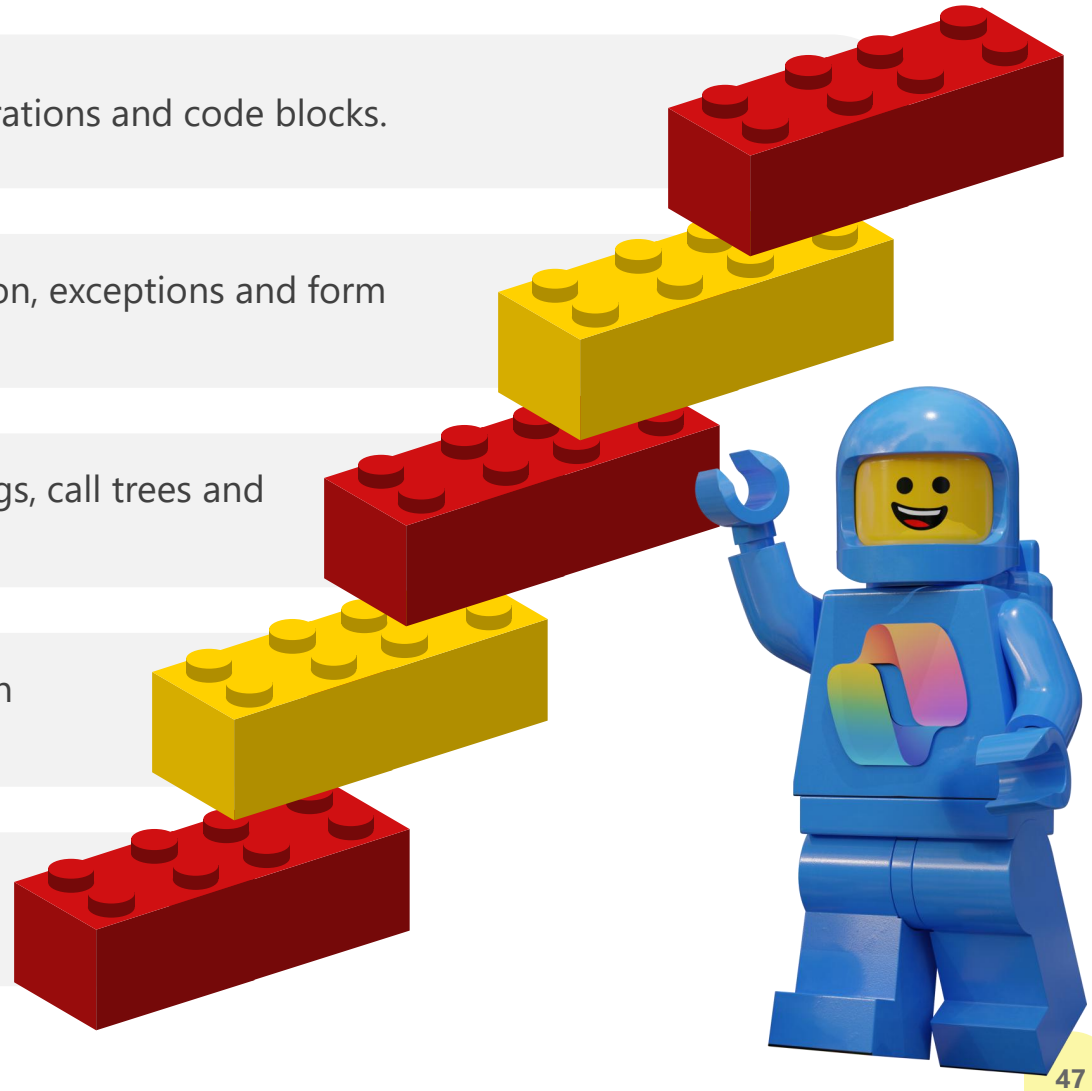
Or your roles and XDS

Load

Or your config, batch and integration load

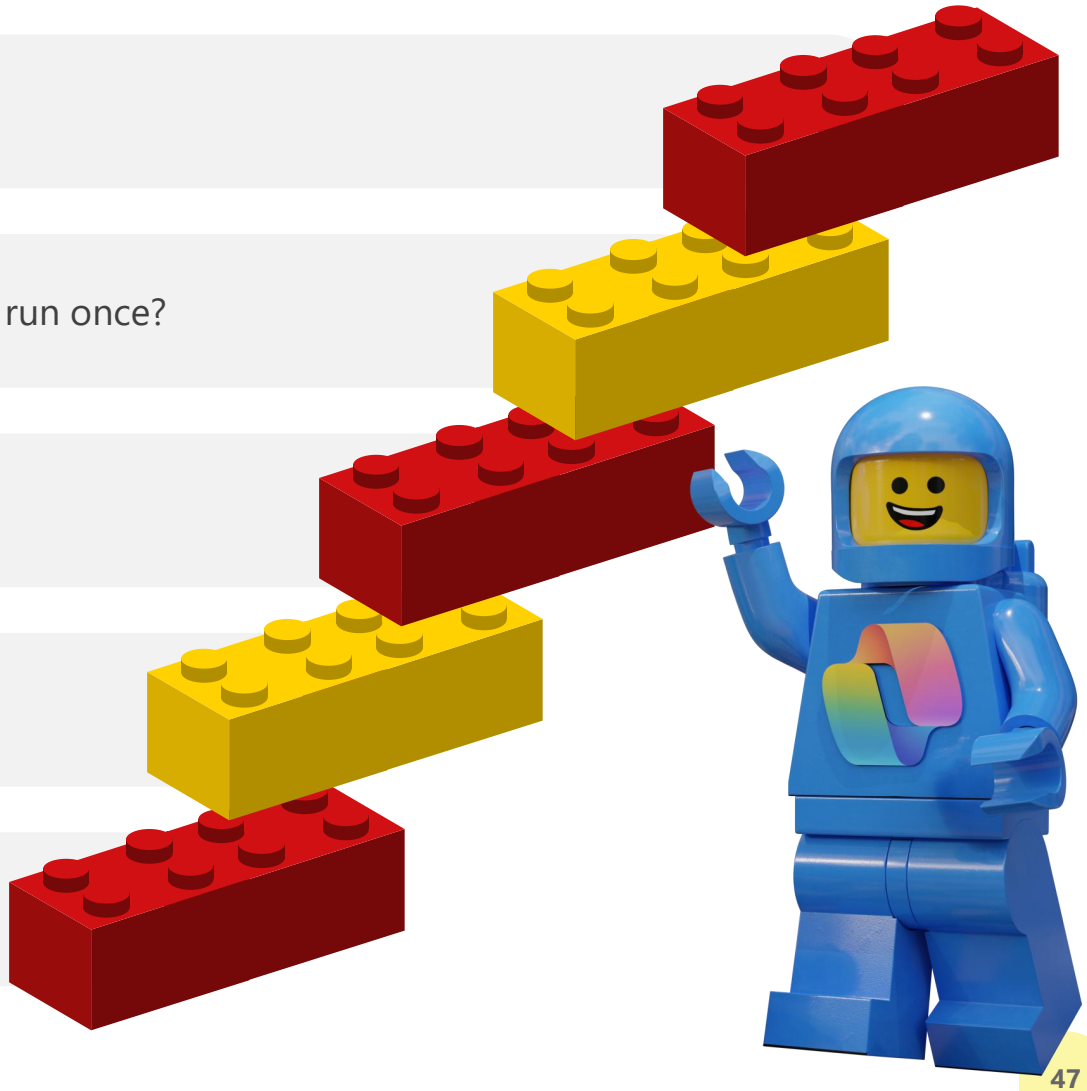
Performance toolbox

PerfTimer	Measure execution time of specific X++ operations and code blocks.
App Insights	Monitor application telemetry, batch execution, exceptions and form usage patterns.
Trace Parser	Analyze X++ execution traces, method timings, call trees and database calls.
Query Store	Identify expensive queries, examine execution plans and compare performance over time.
BP/CAR	Detect potentially inefficient coding patterns before deployment.



Before go-live, ask

Data	Are we retrieving more data than we need?
Repetition	Does something run once per row that could run once?
Testing	Tested with real volume, users and roles?
Config	Can configuration change how this runs?
Ownership	Who owns this data and cleans it?



Questions?

Feedback

Slow is a design choice: F&O decisions you make before go-live

POWER COMMUNITY

DYNAMICS MINDS

Dynamics 365 and Power Platform Community Conference

XPLUS GLOBAL

ARQUICONSULT
Your success. Our solutions.

DOCENTRIC

365 ENGINEERING

POWER PLATFORM
CUSTOMER EXPERIENCE
SALES
FINANCE
SUPPLY CHAIN
COPILOTS
H.R.
DEVOPS
AZURE
SERVICE OPERATIONS

Sponsors

THANK YOU!

POWER COMMUNITY

