THE HIDDEN

SUPERPOWERS OF IBM i

Features You Didn't Know You Already Had

Zero New Licenses.

Db2 for i — time travel, masking, MQTs, JSON, GENERATE_SYSLOG
Query Supervisor + Index Advisor — runtime SQL governance
Cross-System SQL + Ardpgm— federate across IBM i, Oracle, Postgres
Journal Magic — built-in CDC engine, queryable via SQL
Authority Collection + Exit Points — least-privilege from data
Column Encryption + SIEM Logger — GDPR/DORA, all native
DB2 from SSH + PDI in Navigator — modern dev experience
Polyglot Persistence — MariaDB, PostgreSQL, Redis on PASE
ActiveMQ + Service Commander — enterprise messaging, systemd-style ops
Open Source + OCR + MCP + Python — Tesseract, ibmi-mcp, Excel from Db2
Ghostscript + Infoprint — PDF with stapling, duplex, finishing
OmniFind + Geospatial + VROOM — search, ST_Distance, routing
JasperReports + Superset + Zeebe + WebGL + WOPASE — modernization
Bonus: IoT Historian • Vector Store • Anomaly AI

Common Europe IBM i Conference • 2026

Koen Decorte • CD-Invest nv

YOUR GUIDE TODAY

Koen Decorte

Senior IBM i Architect • CD-Invest nv • Melsele, Belgium

- CD-Invest nv — Belgian IBM i consultancy since 1992
- ERP, CRM, Accounting, Document management on IBM i
- PEPPOL Access Point PBE000833
- Stratum Code Intelligence Platform
- Multiple Anthropic / IBM Certifications, IBM Champion and CEAC member

01 SQL + Db2 Core
QSYS2, time travel, masks, MQTs, JSON

02 Query Governance
Query Supervisor + Index Advisor

03 Journal + Objects
Built-in CDC, OS objects via SQL

04 Cross-DB Federation
DRDA + ArdGate to any database

05 Audit + Auth Trace
RCAC + STRAUTCOL = facts, not guesses

06 Exit + Encryption
OS-level hooks + ENCRYPT_AES

07 SIEM Logger
GENERATE_SYSLOG → QRadar / Splunk

08 DB2 CLI + PDI
Interactive Db2 CLI + free APM

09 Open Source + AI
Python, Node, Tesseract, ibmi-mcp

10 Polyglot Stack
MariaDB / Postgres / Redis + JMS + sc

11 Reports + Print
Python→Excel, Ghostscript, Infoprint

12 Search + Geo
OmniFind, ST_Distance, VROOM

13 Modernization
Jasper, Superset, Zeebe, WOPASE

14 Bonus + Swans
IoT, Vectors, AI, OTel, SQLGlot

80%

of IBM i's most powerful features
have never been used
in most shops.

They're not behind a paywall.

They're not in an optional add-on.

They're not in a future release.

They're on your machine.
Right now. Waiting.

Today we open the vault.

SECTION 01

SQL Services

*SELECT * FROM your operating system — QSYS2 makes it possible*

SQL SERVICES

The OS as a Database

Imagine if every CL command you've ever run — WRKACTJOB, WRKDSKSTS, NETSTAT — instead of painting a green screen, returned a result set you could filter, aggregate, export, and schedule.

That's not a future feature. That's QSYS2. 200+ views and table functions, everything IBM i knows about itself, queryable right now.

Run the first one today. You'll never open WRKSYSACT again.

THE PROP

```
SELECT *
FROM QSYS2.ACTIVE_JOB_INFO
ORDER BY
  ELAPSED_CPU_PERCENTAGE
  DESC
FETCH FIRST 10 ROWS ONLY;
```

SQL SERVICES

CL Commands vs SQL Services

SQL returns data you can pipeline — CL commands return screens you must read

Task	Old Way	Superpower
CPU hogs	`WRKSYSACT (screen)`	`SELECT * FROM QSYS2.ACTIVE_JOB_INFO ORDER BY ELAPSED_CPU_PERCENTAGE DESC`
Disk space	`WRKDSKSTS`	`SELECT * FROM QSYS2.DISK_STATISTICS`
Object locks	`WRKOBJELCK`	`SELECT * FROM QSYS2.OBJ_LOCK_INFO`
Slow SQL	`STRDBMON → spool`	`SELECT * FROM QSYS2.SQL_PLAN_CACHE_STATS ORDER BY AVERAGE_ELAPSED_TIME_MS DESC`
TCP ports	`NETSTAT screens`	`SELECT * FROM QSYS2.NETSTAT_INFO WHERE CONNECTION_STATE='LISTEN'`
User audit	`DSPUSRPRF one-by-one`	`SELECT * FROM QSYS2.USER_INFO WHERE PREVIOUS_SIGNON < CURRENT_DATE-90 DAYS`

SECTION 02

Db2 for i

The most advanced built-in database on any platform — used at 10% by most shops

DB2 FOR I — TIME TRAVEL

FOR SYSTEM_TIME AS OF — Query the Past

A client calls: "What was the price of sofa model ST-400 on the 14th of January?" In most shops, this answer requires digging through change logs, calling someone who might remember, or admitting you don't know.

With temporal tables, IBM i recorded the answer automatically. No triggers. No shadow tables. One PERIOD clause in CREATE TABLE and Db2 keeps the full history for you.

This is time travel. It already runs on your machine.

THE PROP

```
FOR SYSTEM_TIME AS OF
  TIMESTAMP('2025-01-14')
-- What did this row look like then?
SELECT ARTNO, LISTPRICE
FROM JORI.JRARTIK
FOR SYSTEM_TIME AS OF
  TIMESTAMP('2025-01-14')
WHERE ARTNO = 'ST-400';
```

DB2 FOR I

Column Masks & Row Permissions — GDPR Built-In

Same table, same query — different users see different data. Zero application code.

CREATE COLUMN MASK MASK_NATL_ID ON CUSTMAST(NATL_ID_NR) — transparent masking at engine level

- VERIFY_GROUP_FOR_USER() — authorised users see the full value, others see ***
- CREATE PERMISSION RESTRICT_ROWS — WHERE SALESREP = SESSION_USER, enforced by engine
- ACTIVATE ROW ACCESS CONTROL AND COLUMN ACCESS CONTROL — one command
- GDPR Article 25 (privacy by design): Db2 enforces it at engine level — every app, every tool, every query
- QSYS2.SYSROWPERMS + QSYS2.SYSCOLUMNS2 — query which tables have protections active

DB2 FOR I

MQTs and Native JSON — Two Hidden Gems

Pre-computed aggregates and structured JSON without any external tool

- MQT: CREATE TABLE summary AS (SELECT GROUP BY…) MAINTAINED BY SYSTEM — auto-refreshed by Db2
- ENABLE QUERY OPTIMIZATION: the optimizer uses your MQT transparently — existing queries speed up without code changes
- JORI example: sofa-sales-by-rep MQT refreshed nightly → dashboard sub-second instead of 45 seconds
- JSON_OBJECT('id' VALUE CUSTNO, 'name' VALUE CUSTNAM) — build REST-ready JSON inside a SELECT statement
- JSON_TABLE() — shred incoming JSON payloads into rows. Parse PEPPOL webhook responses directly in SQL
- Combine: MQT for speed + JSON_OBJECT for API output → your IBM i becomes a native JSON API backend

SECTION 03

Query Supervisor

A bouncer for runaway SQL — Db2 watches, your exit program decides

QUERY SUPERVISOR

The Watchdog That Bites

Every IBM i shop has war stories — the 2 AM call when a single bad SQL statement burns through CPU and brings prod to its knees. The Query Governor (since V3R1) predicts query cost before it runs and is famously inaccurate.

The Query Supervisor (since 2021, IBM i 7.4 TR4 / 7.3 TR10) does the opposite: it watches actual resource consumption while the query runs. When elapsed time, CPU, temp storage, or I/O cross your threshold, Db2 calls your exit program. The program logs, alerts, or cancels.

Almost nobody uses this. Yours can be the shop that does.

THE PROP

```
-- Add a threshold + exit program
CALL QSYS2.ADD_QUERY_THRESHOLD(
  THRESHOLD => 'ELAPSED_TIME',
  COMPARISON_OPERATOR => '>',
  VALUE => 300,
  EXIT_PROGRAM_NAME =>
    'CDSEC/KILLPGM',
  LONG_COMMENT =>
    'Stop runaway SQL');

-- Exit program receives job name,
-- SQL text, resource snapshot.
-- Logs to journal or ENDJOBs.
```

SECTION 04

Index Advisor

Db2 keeps a list of every index it wished it had — plus Encoded Vector Indexes

INDEX ADVISOR

The Engine Has Been Taking Notes

Most shops guess at index strategy. They look at DDS, they look at WHERE clauses, they make educated bets. Meanwhile, since IBM i 6.1, Db2 has been quietly logging every single query where the optimizer thought "I really could have used an index on these columns" — into QSYS2.SYSIXADV.

The engine knows. Sorted by impact. One SQL query gives you the prioritized backlog of indexes to create, ranked by how many times each would have helped and how much time each would have saved.

Bonus: for low-cardinality columns (status, country, gender), switch to Encoded Vector Indexes. The symbol table doubles as a statistics cache — often 10–100× faster than a radix tree for aggregations.

THE PROP

```
-- Db2's prioritized backlog:
SELECT TABLE_NAME,
       KEY_COLUMNS_ADVISED,
       INDEX_TYPE,
       TIMES_ADVISED,
       MTI_USED
FROM   QSYS2.SYSIXADV
WHERE  TABLE_SCHEMA = 'JORI'
ORDER BY TIMES_ADVISED DESC
FETCH FIRST 20 ROWS ONLY;

-- For low-cardinality columns:
CREATE ENCODED VECTOR INDEX
  JORI.IXSTATUS
  ON JORI.JRARTIK (STATUS);
```

SECTION 05

Journal Magic

Your always-on change-data-capture engine — never queried by most shops

JOURNAL MAGIC

IBM i Has Been Recording Every Data Change Since Day 1

Every INSERT. Every UPDATE. Every DELETE. Before images AND after images. Timestamped to the microsecond. Queryable via SQL. Already running on your machine.

Vendors charge a lot per year for CDC products that do exactly this. The difference: you never knew the query.

Here is the query.

THE QUERY

```
-- Every change to CUSTMAST in last
24h
SELECT ENTRY_TIMESTAMP,
       JOURNAL_ENTRY_TYPE,
       JOB_NAME, USER_NAME
FROM TABLE(
  QSYS2.DISPLAY_JOURNAL(
    JOURNAL_LIBRARY=>'CDACCLIB',
    JOURNAL_NAME=>'CUSTJRN',
    STARTING_TIMESTAMP =>
      CURRENT_TIMESTAMP - 24 HOURS
  )
) WHERE OBJECT_NAME = 'CUSTMAST'
ORDER BY ENTRY_TIMESTAMP DESC;
```

JOURNAL MAGIC

Journal as CDC: Feed Any Downstream System

Use IBM i journal entries to drive real-time data pipelines

- Python consumer: poll DISPLAY_JOURNAL for changes since last timestamp — no agent, no middleware needed
- Write changes to data queue, REST webhook, or message broker — all from PASE Python in 30 lines
- CD-Invest PEPPOL: new invoice row detected via journal → triggers Domibus AS4 send automatically
- Remote journal: built-in IBM i replication to secondary IBM i or SQL Server — zero extra product
- QSYS2.JOURNAL_INFO: query journal sizes, receiver chain, replication lag — all in SQL
- Receiver management: ATTACH_JOURNAL, size thresholds, detach — all via SQL procedures

SECTION 06

System Objects

Programs, jobs, data areas, locks, message queues — manage everything via SQL

SYSTEM OBJECTS

Object Metadata: Your Technical Debt Radar

QSYS2.OBJECT_STATISTICS — everything IBM i knows about itself, in a SELECT statement

- SELECT OBJNAME, LAST_USED_TIMESTAMP, OBJ_SIZE FROM QSYS2.OBJECT_STATISTICS WHERE OBJTYPE='*PGM' — find programs not run in 2 years
- Programs unused for 2+ years: your decommission candidates list — done in SQL, not grep
- QSYS2.PROGRAM_INFO + QSYS2.MODULE_INFO: which service programs does each program bind? Complete dependency map
- QSYS2.SQL_ALIASES — find every alias: know your technical landscape before a migration
- QSYS2.SYSROUTINES: every stored procedure, UDF, trigger — the undocumented API surface of your IBM i
- QSYS2.SYSVARIABLES: column-level last-change — know your biggest and busiest tables

SYSTEM OBJECTS

Data Areas, Locks, and Message Queues via SQL

QSYS2 reaches every corner of the IBM i object model

- SELECT * FROM TABLE(QSYS2.DATA_AREA_INFO('CTLFLG','MYLIB')) — read a data area without CL
- QSYS2.OBJ_LOCK_INFO — find locked jobs without WRKOBJELCK; feed into monitoring dashboards
- CALL QSYS2.SEND_MESSAGE: send to QSYSOPR from a SQL procedure — no CL wrapper needed
- QSYS2.OUTPUT_QUEUE_ENTRIES — manage all spooled files via SQL; archive, delete, route from SQL
- QSYS2.USER_SPACE_INFO: peek inside user spaces via SQL; replace hand-rolled CL inspection code
- Combine: build a complete IBM i health dashboard in Apache Superset from QSYS2 data alone

SECTION 07

Cross-System SQL

JOIN tables across two IBM i partitions in one SQL statement — DRDA has been there since the AS/400

CROSS-SYSTEM SQL

Two Systems, One SELECT

Every IBM i shop runs more than one partition. Prod, test, DR, branch offices. Most of them copy data between systems with FTP, save/restore, or a nightly batch job they wrote in 1998 and nobody dares touch.

Db2 for i has spoken DRDA — Distributed Relational Database Architecture — since the 1980s. Define a remote database entry once with WRKRDBDIRE, and a fully-qualified 3-part name (SYSTEM.SCHEMA.TABLE) reaches across the network. The optimiser pushes predicates to the remote node; only matching rows come back.

Compare prod vs test in one query. Join customer data on PROD with archives on DR. Build federated dashboards. All in SQL. Zero ETL.

THE PROP

```
-- One-time setup
ADDRDBDIRE RDB(PRODSYS)
  RMTLOCNAME('prod.cdinvest.be')

-- Compare prod vs test —
-- ONE query, ONE roundtrip
SELECT 'PROD' AS S, COUNT(*)
FROM   PRODSYS.JORI.JRARTIK
UNION ALL
SELECT 'TEST', COUNT(*)
FROM   TESTSYS.JORI.JRARTIK;

-- 3-part naming JOINs too:
-- PRODSYS.JORI.X to TESTSYS.JORI.Y
```

SECTION 08

ARDPGM

*Query Oracle, MySQL, SQL Server, Postgres from STRSQL or embedded RPG — via *ARDPGM*

ARDGATE

When DRDA Doesn't Reach

Cross-System SQL works between IBM i partitions because they all speak DRDA. But what about Oracle? MySQL? SQL Server? PostgreSQL? Most shops still build COBOL/RPG → IFS-CSV → import-on-the-other-side pipelines for this.

Db2 for i has another exit point almost nobody knows: the SQL Client Integration Exit, or *ARDPGM. When a remote-database-directory entry points at an *ARDPGM, every SQL call to that RDB is routed to your exit program. Dieter Bender's open-source ArdGate is that program — a small RPG stub plus a long-running JVM that bridges Db2 ↔ JDBC.

ADDRDBDIRE once. Drop the JDBC driver JAR in /jvagate/lib. CONNECT TO ORACLEDB. From STRSQL, from QMQRY, or from RPG with embedded SQL — same syntax, different planet.

THE PROP

```
/* One-time setup */
ADDRDBDIRE RDB(ORACLEDB)
  RMTLOCNAME(ARDPGM)
  ARDPGM(JVAGATE/JDBCGATE)

/* global.properties: */
ard.url.ORACLEDB=
  jdbc:oracle:thin:@host:1521:xe

/* From STRSQL or RPG: */
CONNECT TO ORACLEDB;
SELECT * FROM HR.EMPLOYEES
FETCH FIRST 10 ROWS ONLY;
```

SECTION 09

Security & Audit

Column masks, QAUDJRN, exit programs — enterprise security without enterprise spend

SECURITY & AUDIT

Authority by SQL + QAUDJRN Analysis

IBM i security is the strongest in the industry — but only if you use it

- QSYS2.USER_INFO: last signon, password expiry, special authorities — your quarterly user audit in one SELECT
- QSYS2.AUTHORITY_COLLECTION: track every object access — who read what, when
- QAUDJRN *SECURITY: every password failure, special authority use, object ownership change — queryable journal
- QSYS2.NETSTAT_INFO: find unexpected listening ports — security posture check via SQL
- Exit programs (QIBM_QSO_ACCEPT): intercept every FTP, ODBC, DDM connection — granular access control
- Pattern: export QAUDJRN entries to Db2 daily → dashboard in Apache Superset → detect anomalies with Kafka

SECTION 10

Authority Collection

Find the actual minimum permissions every user needs — facts, not guesses

AUTHORITY COLLECTION

End the *ALLOBJ Era — With Data

Every IBM i shop has *ALLOBJ users "because we don't know what they really need to access." The real reason: we're afraid to lock anything down because the app might break in production three weeks from now.

STRAUTCOL records every authority check the system performs — what was requested, what was granted, by whom, against which object, in which job. Let the application run for a week. Query QSYS2.AUTHORITY_COLLECTION. Adopt least privilege based on facts.

Available since 7.3. New in 7.5 TR6 / 7.6: per-IFS-path tracking via QSYS2.AUTHORITY_COLLECTION_IFS.

THE PROP

```
-- Start collecting on the app user
STRAUTCOL USRPRF(JORIAPP)
          OBJ((*ALL *ALL *ALL))
          INFTYPE(*ALL)

-- After a week of production:
SELECT OBJECT_NAME,
       OBJECT_LIBRARY,
       REQUIRED_AUTHORITY,
       COUNT(*) AS HITS
FROM   QSYS2.AUTHORITY_COLLECTION
WHERE  USER_NAME = 'JORIAPP'
GROUP BY 1,2,3
ORDER BY HITS DESC;

ENDAUTCOL USRPRF(JORIAPP)
```

SECTION 11

Exit Points

Plug your code into every OS boundary — FTP, ODBC, signon, command, PWRDWNSYS

EXIT POINTS

Your Code, At Every OS Doorway

Commercial "IBM i exit-point firewall" products start at €15k/year. They have nice dashboards. Underneath, they ship one CL command: ADDEXITPGM. The OS has been exit-point-driven since V3R6.

Run WRKREGINF on any IBM i and scroll. Over 100 exit points: QIBM_QSO_ACCEPT (every inbound TCP), QIBM_QZSO_SIGNONSRV (every signon), QIBM_QSY_VLD_PASSWRD (password validation), QIBM_QPWFS_FILE_SERV (file server), QIBM_QWC_PWRDWNSYS (shutdown).

Want to reject FTP from foreign IPs? Exit program. Want to log every ODBC connection? Exit program. Want a confirmation before PWRDWNSYS? Exit program.

THE PROP

```
/* Reject inbound TCP from non-EU
IPs
   QIBM_QSO_ACCEPT runs BEFORE the
OS
   accepts the socket — your code
   gets the final vote. */

ADDEXITPGM
  EXITPNT(QIBM_QSO_ACCEPT)
  FORMAT(ACPT0100)
  PGMNBR(1)
  PGM(CDSEC/SOACCEPT)
  TEXT('Geo-fence inbound TCP')

/* List every exit point registered
*/
WRKREGINF
```

SECTION 12

Column Encryption

ENCRYPT_AES / DECRYPT_CHAR — PII at rest, no third-party product

COLUMN ENCRYPTION

The Crypto Library Inside Your SELECT

GDPR Article 32. DORA Article 9. Both ask the same question: "Is personally identifiable data encrypted at rest?" Most shops answer by buying a third-party product, by storing PII in a separate masked column, or by hoping the auditor doesn't ask too hard.

Db2 for i ships ENCRYPT_AES() and DECRYPT_CHAR() as native SQL scalar functions. Encryption happens in the engine. The session sets the password. Authorised callers decrypt; everyone else sees ciphertext.

Pair with the column masks slide (RCAC) — mask at the engine, encrypt at the column. Defence in depth, two layers, zero new licences.

THE PROP

```
-- Set encryption password for
session
-- (production: fetch from a key
vault)
SET ENCRYPTION PASSWORD = :USERKEY;

-- Store the Belgian national ID
-- encrypted at rest
INSERT INTO CUSTMAST
  (CUSTNO, NATL_ID_NR)
VALUES (1001,
  ENCRYPT_AES('BE 12.34.56-
789.01'));

-- Authorised callers see plaintext
SELECT CUSTNO,
  DECRYPT_CHAR(NATL_ID_NR) AS
NATL_ID
FROM   CUSTMAST
WHERE  CUSTNO = 1001;
```

SECTION 13

SIEM Logger

Turn IBM i into its own security monitor — Db2 + syslog + SQL service

SIEM LOGGER

Your IBM i Sees Everything. Nobody Is Listening.

Every failed login. Every job abend. Every authority escalation. IBM i journals all of it — and has since V2R2. It sits in QAUDJRN and QHST, seen by nobody.

A SIEM (Security Information and Event Management) system costs €80k+/year and expects RFC5424-formatted syslog events. IBM i has been able to generate exactly that since 2017 — via a single SQL parameter nobody told you about.

One SQL call turns your QAUDJRN into a live RFC5424 syslog feed. syslogd is already in PASE Option 33. The logger command ships each event to QRadar, Splunk, or Elastic.

Zero new software. Zero new licences.

THE PROP

```
-- GENERATE_SYSLOG turns QAUDJRN
into
-- RFC5424/CEF format in one SQL
call
SELECT
  syslog_facility,
  syslog_severity,
  cast(syslog_event AS
    varchar(2048) CCSID 37)
FROM TABLE(
  QSYS2.DISPLAY_JOURNAL(
    'QSYS', 'QAUDJRN',
    GENERATE_SYSLOG => 'RFC5424'
  )
) AS X
WHERE syslog_event IS NOT NULL;
```

SIEM LOGGER

QAUDJRN + GENERATE_SYSLOG + syslogd: The Real Pipeline

RFC5424/CEF from SQL → PASE logger → syslogd → QRadar / Splunk / Elastic — all in Option 33

```
-- ── STEP 1: SQL generates RFC5424 events from QAUDJRN ──────────
-- GENERATE_SYSLOG available since IBM i 7.2 PTF SF99702 level 19
SELECT cast(syslog_event AS varchar(2048) CCSID 37) AS EVT
FROM TABLE(QSYS2.DISPLAY_JOURNAL('QSYS','QAUDJRN',
    GENERATE_SYSLOG => 'RFC5424',
    EOF_DELAY => 30)        -- stream indefinitely, 30s poll
) AS X WHERE syslog_event IS NOT NULL;
-- Output: <11>1 2025-05-01T08:12:03.000-06:00 MY-IBMI - - - -
--         CEF:0|IBM|IBM i|7.5|QSYS-AUDJRN|T=PW|Low|...

-- ── STEP 2: PASE logger ships each event to syslogd ────────────
-- syslogd is part of 5770SS1 Option 33 — already on your machine
-- Start it once: SBMJOB CMD(STRQSH CMD(
--   '/QOpenSys/usr/sbin/syslogd')) JOB(SYSLOGD)
-- Shell: echo "$EVT" | logger -p auth.warning

-- ── STEP 3: /QOpenSys/etc/syslog.conf routes to your SIEM ──────
-- auth.warning  @siem.cdinvest.be:514   (UDP to QRadar/Splunk)
```

💡 *GENERATE_SYSLOG + EOF_DELAY = a live RFC5424 stream from QAUDJRN to any SIEM. Since 2017. Free.*

SECTION 14

DB2 from SSH

One command. Any terminal. Your entire database.

DB2 FROM SSH

Your Db2 Has a CLI. You've Just Never Opened It.

Every DBA on PostgreSQL knows psql. Every MySQL DBA knows mysql. Every Oracle DBA knows sqlplus.

IBM i has had its own interactive SQL CLI since forever — the db2 command in PASE. SSH into your IBM i, type db2, press Enter. You have a full interactive Db2 session. Tab completion. History. Scripting.

No 5250. No ACS. No JDBC. Just SSH and your data.

THE PROP

```
$ ssh koen@my-ibmi
$ db2
db2 => CONNECT TO *LOCAL
db2 => SELECT ARTNO, LISTPRICE
       FROM JORI.JRARTIK
       WHERE ARTFAM = 'SOFA'
       FETCH FIRST 5 ROWS ONLY;
db2 => QUIT

# Or non-interactive:
$ db2 "SELECT COUNT(*) FROM
QSYS2.ACTIVE_JOB_INFO"
$ db2 -f /home/koen/myquery.sql
```

DB2 FROM SSH

Scripting, Pipelines and Automation via db2 CLI

SSH + db2 command = full Db2 access from any CI/CD pipeline, shell script, or monitoring tool

```
# ── Interactive session ──────────────────────────────
$ ssh admin@my-ibmi.cdinvest.be
$ db2              # opens interactive Db2 CLI
db2 => SET SCHEMA CDACCLIB
db2 => SELECT * FROM QSYS2.ACTIVE_JOB_INFO ORDER BY 1 DESC
db2 => QUIT

# ── Non-interactive: pipe into shell scripts ─────────
$ db2 -x "SELECT ARTNO,LISTPRICE FROM JORI.JRARTIK" > prices.csv

# ── Run a SQL file ───────────────────────────────────
$ db2 -f daily_audit.sql | mail -s 'IBM i Daily Audit' ops@cdinvest.be

# ── From a CI/CD pipeline (GitHub Actions, Jenkins) ──
$ ssh ibmi 'db2 -x "SELECT COUNT(*) FROM CDACCLIB.INVOICES"'

# ── Use CLPPLUS for Oracle-style scripting syntax ────
$ clpplus MYUSER/***@my-ibmi:446/MYDB
```

💡 *db2 + SSH = your IBM i is now scriptable from any DevOps toolchain. No ODBC. No JDBC. No middleware.*

SECTION 15

PDI in Navigator

Performance Data Investigator — the free APM that ships with the OS

PDI IN NAVIGATOR

Datadog Is Already On Your Machine

Collection Services has been recording CPU, memory, disk, jobs, SQL plan cache, and wait-time breakdowns at second-level granularity for decades. PDI, built into the new Navigator for i, visualises all of it: flame graphs, wait-time charts, top SQL by elapsed time, top jobs by CPU, top consumers by I/O.

The "should we buy an APM?" meeting ends here. You already have one. It has been collecting data since the day you IPL'd.

Open it. Today.

THE PROP

```
# Open PDI in your browser:
https://your-ibmi:2005/Navigator
  → Performance
  → Investigate Data
  → CPU Utilisation by Thread

-- Same data, queryable from SQL:
SELECT QUERY_TIME_ELAPSED_MS,
       USER_NAME, JOB_NAME,
       SUBSTR(STATEMENT_TEXT, 1, 80)
FROM   QSYS2.SQL_PLAN_CACHE_TOP
ORDER BY QUERY_TIME_ELAPSED_MS DESC
FETCH FIRST 10 ROWS ONLY;
```

SECTION 16

Open Source + OCR + MCP

Python, Node, Git, Tesseract OCR and the IBM i MCP server — all via yum

OPEN SOURCE

The Open-Source Stack: Three Commands Away

IBM i Open Source Catalog — yum install and you're running

- yum install python39 python39-pip python39-ibm_db — Python 3.9 + DB2 driver: production-ready
- yum install nodejs npm git — Node.js, npm, git in PASE: modern development stack on your IBM i
- yum install tesseract — Google's OCR engine, 100+ languages: runs in PASE natively without cloud
- yum install boost-filesystem convos — ILEastic dependencies for REST APIs without Apache
- Production example: 300 IFS TIFF files → Tesseract → text extraction → INSERT INTO Db2: zero manual work

OPEN SOURCE

Tesseract OCR on IBM i: Scan → Db2 in 30 Lines

Open-source OCR engine running natively on IBM i PASE — invoice scanning without cloud

```
# yum install tesseract  ← Google's OCR engine, 100+ languages, runs in PASE
import pytesseract, ibm_db_dbi, glob
from PIL import Image

conn = ibm_db_dbi.connect('*LOCAL', 'OCRUSER', '***')
cur  = conn.cursor()

for tiff in glob.glob('/ifs/invoices/*.tif'):
    text = pytesseract.image_to_string(Image.open(tiff),
                                       lang='fra+nld')
    # Extract invoice number from recognised text
    inv_no = extract_inv_no(text)   # your regex here
    cur.execute(
        'INSERT INTO MYLIB.OCR_RESULTS VALUES (?,?,?)',
        (tiff, inv_no, text)
    )
conn.commit()
# Quality tip: 300+ DPI scans → OCR accuracy above 99%
```

💡 *Pipeline: IFS → Tesseract → extract fields → Db2 → your ERP. Zero cloud. Zero licence.*

OPEN SOURCE + MCP

IBM i MCP Server: AI Talks Directly to Your Data

IBM released an IBM i MCP server beta (October 2025) — any AI tool can now query your Db2

- MCP = Model Context Protocol — the USB-C of AI tool integration (Anthropic open standard)
- github.com/IBM/ibmi-mcp — IBM's official IBM i MCP server beta release
- Claude, Cursor, VS Code Copilot: point MCP at your IBM i — ask questions, get SQL queries, navigate schemas
- CD-Invest JORI example: Claude queries JRARTIK and answers 'Which sofas sold best in Q1?' directly from Db2
- MCP donated to Linux Foundation / AAIF — permanent open standard, not vendor lock-in
- AI-Assisted RPG Development: Claude reads your QRPGLESRC members, suggests refactors, generates documentation

SECTION 17

Polyglot Persistence

MariaDB, PostgreSQL, Redis — three more database engines, same IBM i partition

POLYGLOT PERSISTENCE

Three Databases. One LPAR.

Your IBM i is already running Db2. What most shops don't realize: yum on PASE installs three more production-grade database engines, all running side-by-side on the same Power LPAR.

MariaDB 10.6 — the open-source MySQL successor — handles the WordPress site marketing keeps asking for, or the e-commerce backend a new partner brings. PostgreSQL 12 handles JSON-heavy workloads and modern microservice schemas your Db2 doesn't need to learn. Redis 6 sits in front of Db2 as a microsecond-latency cache for hot reads.

All three: backed up by your existing strategy, monitored by your existing tooling, secured by your existing IBM i users. Polyglot persistence — without provisioning a single new server.

THE STACK

```
# All three from yum:
yum install mariadb-10.6-server
yum install postgresql12-server
yum install redis

# Initialize and start MariaDB
mysql_install_db --user=qsecofr
mysqld_safe &

# Initialize and start Postgres
initdb -D /pgsql/data
pg_ctl -D /pgsql/data start

# Start Redis
redis-server &

# Connect from any client
# on the same machine or LAN
```

SECTION 18

ActiveMQ

Enterprise message broker on IBM i — JMS, AMQP, STOMP, MQTT — yum install activemq

ACTIVEMQ

Your Broker, Same Box as the Database It Feeds

Every IBM i modernization project hits the same line item: "add a message queue between us and the new system." Most shops provision a separate Linux VM running RabbitMQ or ActiveMQ, then route everything through it — extra hop, extra hardware, extra failure mode.

The IBM i Open Source Catalog ships ActiveMQ 5.18 directly. One yum install, one PASE process. JMS, AMQP, STOMP, MQTT — every modern messaging protocol, one broker, on the same partition as your Db2.

Microservices queue orders. IoT devices publish telemetry. Mobile apps subscribe to notifications. Java apps consume JMS. Python apps publish via STOMP. All talking to your IBM i without a single hop through external infrastructure.

THE PROP

```
# Install + start
yum install activemq
sc start activemq

# Web console at:
# http://your-ibmi:8161/admin
# (admin/admin by default)

# Publish from Python:
import stomp
c = stomp.Connection(
  [('localhost', 61613)])
c.connect('admin','admin')
c.send(
  body='order-12345',
  destination='/queue/ORDERS')

# Consume from RPG or Java
```

SECTION 19

Service Commander

systemd-style service management for PASE — sc start, sc enable, sc status

SERVICE COMMANDER

systemd, But For Your IBM i

Every Linux admin in the world runs systemctl start nginx, systemctl enable redis, systemctl status all. IBM i PASE has never had that — every shop wrote its own SBMJOB / ENDJOB / scinit / batch-immediate-job ritual, every open-source server installed via yum had a different startup style, and "ps -ef | grep" was the operational manual.

Service Commander (yum install service-commander) brings the same systemd-style discipline to PASE. Define a service in a small YAML file. Start, stop, enable, disable, check status — one command, every service. Auto-start on IPL. Centralized logs.

Pair it with the polyglot stack: sc start mariadb postgresql12 redis activemq nginx — one line, five servers, two seconds.

THE PROP

```
yum install service-commander

# Define a service in YAML:
# /QOpenSys/etc/sc/services/
#   nginx.yml
---
name: nginx
start_cmd:
  /QOpenSys/pkgs/sbin/nginx
stop_cmd:
  /QOpenSys/pkgs/sbin/nginx -s stop
check_alive: PORT
check_alive_criteria: 80

# Then:
sc start  nginx
sc enable nginx     # auto-IPL
sc status all
```

SECTION 20

Python → Excel

Db2 to a formatted XLSX — pandas + openpyxl + ibm_db, all from PASE Python

PYTHON → EXCEL

Stop Building Reports in RPG That Land in Excel Anyway

Every business user wants the report in Excel. Every IBM i shop generates the report from RPG, exports to CSV, opens in Excel, formats columns, applies filters, saves as XLSX, emails it. Every. Day.

PASE Python already has pandas, ibm_db, and a one-line install for openpyxl. A 15-line script connects to Db2, runs the query, formats headers, freezes the top row, autosizes columns, adds an autofilter, and writes the file to the IFS where any user can grab it via NetServer.

Schedule it through ADD_JOB_SCHEDULER_ENTRY. The "monthly sales report" meeting is now a notification, not a process.

THE PROP

```
# pip install --user openpyxl
import pandas as pd, ibm_db_dbi

conn = ibm_db_dbi.connect(
  '*LOCAL','RPTUSER','***')

df = pd.read_sql("""
  SELECT ARTNO, ARTNAM, LISTPRICE
  FROM   JORI.JRARTIK
  WHERE  ARTFAM='SOFA'
""", conn)

df.to_excel('/ifs/sofa.xlsx',
  index=False)
```

SECTION 21

Ghostscript

PDF with finishing — stapling, duplex, hole-punching via PJL, all from CL

GHOSTSCRIPT

Stapling, Duplex, Hole-Punching — From CL

Most IBM i shops own modern multi-function printers — Lexmark, HP, Ricoh — that staple, punch, duplex, fold, and saddle-stitch. A PDF file program knows nothing about any of that. The MFP falls back to plain output, and somebody in the warehouse staples 200 invoices by hand every morning.

Ghostscript on PASE (yum install ghostscript) renders PostScript and PDF that includes PJL — Printer Job Language — finishing directives the MFP actually understands. Tell the printer to staple top-left. Duplex long-edge. Punch three holes. Saddle-stitch a 16-page booklet.

Same RPG. Same spool file. The buttons on your printer that nobody has ever pressed finally get pressed — by a CL job.

THE PROP

```
# Spool → PostScript
CPYSPLF FILE(INV) TOFILE(*TOSTMF)
  TOSTMF('/tmp/inv.ps')
WSCST(*PSCOE)

# Render PDF with finishing:
gs -sDEVICE=pdfwrite \
   -sOutputFile=/tmp/inv.pdf \
   -c "<< /Staple 3 \
         /Duplex true >> \
       setpagedevice" \
   -f /tmp/inv.ps

# Print on finishing-capable MFP
lp -d MFPSTAPLE /tmp/inv.pdf
```

SECTION 22

Infoprint Server

Native IBM spool-to-PDF transformer — 5770-IP1, output queue → PDF in IFS, automatically

INFOPRINT SERVER

Spool → PDF, as an Output Queue Property

If Ghostscript is the open-source hammer, Infoprint Server is the IBM-engineered toolset. Licensed program 5770-IP1 takes any spool file — SCS, AFP, line, IPDS — and produces PDF, PostScript, PCL, or AFP, with PCL-XL coming in TR support.

Configure an output queue with PSF/400 or IPS rules, point applications at it, and the spool file appears in the IFS as a PDF with no application changes whatsoever. Combine with IPS email integration and the PDF goes straight to the customer's inbox.

For invoice runs, picking lists, statements, and PEPPOL document attachments, this is the production-grade path. No application code touched.

THE FLOW

```
/* Transform spool → PDF and store
   in the IFS, on an output queue:
   Tools → Infoprint Server
   → Output Queue Rules
   → Add Transform Rule

   FORMAT  : PDF
   TARGET  :
/ifs/invoices/&JOBNAME.pdf
   USE     : *AFPDS or *SCS

   Or from a program: */
CRTAFPDTA  FILE(INV)
           DEVTYPE(*AFPDS)

STMF('/ifs/invoices/x.pdf')
```

SECTION 23

OmniFind

Full-text search engine built into Db2 — install 5733-OMF, call a stored procedure

OMNIFIND

Full-Text Search in Db2 — No Elasticsearch Required

5733-OMF is already available for your IBM i — one install unlocks enterprise search

- SYSPROC.SYSTS_CREATE('MYLIB', 'PRODIDX', 'MYLIB', 'PRODUCTS', '"DESCRIPTION" "COMMENTS"') — index two columns
- SYSPROC.SYSTS_UPDATE('MYLIB', 'PRODIDX') — incremental refresh: only changed rows reindexed
- SELECT * FROM TABLE(MYLIB.PRODIDX_SRCH(QUERY=>'leather sofa Italy NEAR beige', NUMDOCS=>20)) — natural language search
- Scores returned: 0.0–1.0 relevance per row — sort by relevance just like any web search engine
- German, Dutch, French, English: OmniFind handles all with language-specific stemming built in
- JORI use case: 'find all sofas described as Italian leather, nubuck, or alcantara' — one SRCH call, no LIKE wildcards

SECTION 24

Geospatial + VROOM

ST_functions + OSRM route optimisation — geography runs inside Db2

GEOSPATIAL

ST_DISTANCE in Db2 — Find the 5 Nearest Dealers

db2gse geospatial extension included in 5770-SS1 — no licence, no install beyond enabling

```
-- Store dealer locations as geometry in Db2
ALTER TABLE DEALERS ADD COLUMN GEO_POINT DB2GSE.ST_POINT;
UPDATE DEALERS SET GEO_POINT =
  DB2GSE.ST_Point(LONGITUDE, LATITUDE, 4326);

-- Find 5 nearest dealers to any GPS coordinate
SELECT DEALER_NAME, CITY,
  ROUND(DB2GSE.ST_Distance(GEO_POINT,
    DB2GSE.ST_Point(4.3517, 50.8503, 4326), 'KILOMETER'), 1)
  AS KM_FROM_BRUSSELS
FROM DEALERS
ORDER BY KM_FROM_BRUSSELS
FETCH FIRST 5 ROWS ONLY;

-- ST_Within, ST_Intersects, ST_Buffer: full PostGIS-class ops
-- JORI use case: find all dealers within 50km of a trade show
```

💡 *Geospatial IS in your Db2. The function is DB2GSE.ST_Distance. Run it today.*

GEOSPATIAL + VROOM

VROOM Route Optimisation on IBM i

Vehicle Routing Problem solver — runs on PASE, optimises delivery routes in milliseconds

- yum install vroom — open-source VRP solver built on OSRM routing engine: one command
- Input: JSON with vehicles (capacity, start/end location) and jobs (location, time window, size)
- Output: optimised route per vehicle — minimises distance, respects time windows, handles multiple depots
- JORI example: 3 delivery trucks, 47 stops, Belgium + Netherlands — VROOM computes in under 2 seconds
- Python bridge: read stops from Db2 → call VROOM JSON API → write routes back to Db2 → print waybills via JasperReports
- This stack (Db2 + VROOM + Jasper + PASE Python) replaces a routing SaaS product

SECTION 25

Modern Toolchain

JasperReports, Apache Superset, Zeebe, WebGL — enterprise stack via yum

MODERN TOOLCHAIN

JasperReports: Enterprise PDF/Excel Reports from Db2

Pixel-perfect reports — charts, barcodes, subreports — driven directly by Db2 for i

- yum install jasperreports — JasperReports Server available via IBM i Open Source Catalog
- Data source: Db2 for i via JDBC — your existing tables, views, and SQL procedures become report sources
- JORI example: sofa configurator generates a PDF quotation with embedded 3D render, pricing, and barcode
- iText 7: sign PDFs with IBM i certificates — PEPPOL invoice PDFs signed natively on IBM i without Windows
- JasperServer REST API: call from RPG via ILEastic → trigger report generation from any IBM i application
- Output: PDF, XLSX, CSV, HTML, DOCX — same Jasper template, any format

MODERN TOOLCHAIN

Apache Superset + Zeebe + WebGL

A full enterprise analytics and workflow stack — all installable from PASE yum

- Apache Superset: yum install superset — drag-and-drop dashboards directly over Db2 for i via SQLAlchemy
- Zero data copy: Superset queries Db2 live — no ETL, no warehouse, no data movement needed
- JORI Superset dashboard: sales per rep, sofa family, country — refresh every 5 minutes, viewed on TV in warehouse
- Zeebe / Camunda: BPMN workflow engine on PASE — model your IBM i business processes as diagrams, not CL code
- WebGL 3D configurator: JORI's sofa configurator uses Three.js + IBM i REST API to show live 3D renders in a browser
- Full stack: RPG ILE backend → ILEastic REST → Three.js WebGL frontend → runs on IBM i PASE with no middleware

SECTION 26

WOPASE

A native ILE package manager — GitHub to IBM i with zero PASE dependency

WOPASE

A Package Manager That Speaks Native IBM i

yum installs PASE packages — AIX binaries that run in the compatibility layer.

WOPASE by Andrea Ribuoli goes further: it installs native ILE packages from GitHub using nothing but ILE CL and IBM i's built-in HTTPS. No PASE. No Python. No SSH prerequisite. Just IBM i being IBM i.

Think of it as npm or pip, but for ILE objects. A GUIDANCE.TXT in a GitHub repo, a BUILD.CLLE script, and PASERIE/INSTALL — and your code is on the target machine. The installer downloads, compiles, and installs in a batch job. Completely native.

THE INSTALL

```
/* ILE CL — no PASE needed */
CALL PASERIE/INSTALL
  REPO_OWNER('AndreaRibuoli')
  REPOSITORY('WOPASE')
  YOURGITPDA('ghp_yourtoken')

/* That's it. The installer:
   1. Calls GitHub API over HTTPS
   2. Downloads GUIDANCE.TXT
   3. Compiles BUILD.CLLE in QTEMP
   4. Runs the build script
   5. Installs to target library */
```

WOPASE

How WOPASE Works — Native ILE All the Way Down

github.com/AndreaRibuoli/WOPASE — pure ILE CL, zero PASE dependency

- WOPASE uses IBM i's built-in HTTPS client (no curl, no wget) to call the GitHub API natively from ILE CL
- GUIDANCE.TXT in your GitHub repo lists the source members; BUILD.CLLE compiles them on the target system
- WOPASE/INSTALL is the command — give it your repo owner, repo name, and a GitHub token: done
- WOPASE/LIBCLONE: turn an existing IBM i library into a WOPASE-ready GitHub package in one command
- WOPASE/INSTALLOC: test your package locally before pushing to GitHub — development workflow built-in
- The philosophical point: ILE CL is part of the OS (5770SS1) — no 5770WDS required for CL compilation
- Result: an IBM i developer can publish and install ILE programs using only what's already on the machine

SECTION 27

Bonus Powers

IoT Historian • Vector Store • Anomaly Detection AI

BONUS POWERS

IoT Historian on IBM i

IBM i as a time-series database for sensor data — MQTT → Db2, already running

- yum install mosquitto — MQTT broker on PASE: receives sensor data from factory floor, warehouses, machines
- Python MQTT subscriber: paho-mqtt reads temperature, vibration, power data → INSERT INTO Db2 time-series table
- QSYS2.ADD_JOB_SCHEDULER_ENTRY: run anomaly check every minute — no separate IoT platform needed
- Db2 temporal tables: store 5 years of sensor history, query any point-in-time in microseconds
- Dashboard: Apache Superset over the Db2 IoT table — live factory floor KPIs with 30-second refresh
- The cost of MQTT + Python + Db2 on your IBM i: zero.

BONUS POWERS

Vector Store + Anomaly Detection AI

Semantic search and ML anomaly detection — running inside Db2 on IBM i

- Vector embeddings: Python sentence-transformers generates 384-dim vectors from text → stored as BLOB in Db2
- Semantic search: cosine similarity in Python → find 'Italian leather sofas' even when description says 'pelle italiana'
- QSYS2.SQL_PLAN_CACHE_STATS: feed query execution times into scikit-learn IsolationForest → detect slow-query anomalies
- Anomaly detection: train on 30 days of normal CPU/disk/job data → flag deviations before users notice
- Python scikit-learn + ibm_db: train on IBM i, predict on IBM i — no cloud AI service required
- MCP + vector store: Claude queries semantic embeddings of your RPG source code → finds similar programs to a description

SECTION 28

Black Swans

Things that sound impossible until you try them

BLACK SWAN 1 — OpenTelemetry Tracing on IBM i

Distributed tracing from RPG to Grafana — using the OTLP standard

```
# yum install python39-opentelemetry-sdk
from opentelemetry import trace
from opentelemetry.exporter.otlp.proto.grpc import OTLPSpanExporter
from opentelemetry.sdk.trace import TracerProvider

provider = TracerProvider()
exporter = OTLPSpanExporter(endpoint='http://grafana:4317')
provider.add_span_processor(BatchSpanProcessor(exporter))
trace.set_tracer_provider(provider)

tracer = trace.get_tracer('ibmi.cdacclib')
with tracer.start_as_current_span('invoice_processing') as span:
    span.set_attribute('invoice.count', 847)
    # Your Db2 calls, RPG service program calls here
    # → trace flows to Grafana Tempo, Jaeger, or Datadog
```

💡 *IBM i is already the most instrumented platform in enterprise IT — OTLP exposes that to the DevOps world.*

BLACK SWAN 2 — SQLGlot: Parse Every SQL Statement

Audit table usage, detect migration blockers, transpile Db2 to Snowflake — 31 dialects, zero dependencies

```
# pip install sqlglot  — zero dependencies, pure Python
import sqlglot, ibm_db_dbi, re
from sqlglot import exp

# Scan every RPG member for EXEC SQL → build table-usage map
conn = ibm_db_dbi.connect('*LOCAL','USER','***')
cur  = conn.cursor()
cur.execute("SELECT SRCMBR, SRCDTA FROM CDACCLIB.QRPGLESRC"
            "WHERE UPPER(SRCDTA) LIKE '%EXEC SQL%'")
table_map = {}
for pgm, line in cur.fetchall():
    m = re.search(r'EXEC SQL\s+(.*)', line, re.I)
    if m:
        ast = sqlglot.parse_one(m.group(1), dialect='tsql')
        tables = [t.name for t in ast.find_all(exp.Table)]
        table_map.setdefault(pgm, set()).update(tables)
# Result: every program → every table it touches
```

💡 *31 dialects: Db2, Spark, Snowflake, BigQuery, PostgreSQL. Parse EXEC SQL from RPG source in one script.*

BLACK SWAN 3 — NetworkX: Call Graph of Your Entire IBM i

Orphan programs, blast radius, circular dependencies — 40 lines of Python from QSYS2.PROGRAM_INFO

```
# pip install networkx  — pure Python, zero deps
import networkx as nx, ibm_db_dbi
conn = ibm_db_dbi.connect('*LOCAL','NXUSER','***')
cur  = conn.cursor()
cur.execute("""
    SELECT P.PROGRAM_NAME, B.BOUND_SERVICE_PROGRAM_NAME
    FROM   QSYS2.PROGRAM_INFO P
    LEFT   JOIN QSYS2.BOUND_MODULE_INFO B
           ON B.PROGRAM_NAME = P.PROGRAM_NAME
    WHERE  P.PROGRAM_LIBRARY = 'CDACCLIB'""")
G = nx.DiGraph()
for pgm, srvpgm in cur.fetchall():
    if srvpgm: G.add_edge(pgm, srvpgm)
# Q1: Orphaned programs (safe to delete?)
orphans = [n for n in G if G.in_degree(n)==0 and G.out_degree(n)>0]
# Q2: Blast radius — what breaks if ACCUTIL changes?
blast = nx.ancestors(G, 'ACCUTIL')
```

💡 *QSYS2.PROGRAM_INFO already has the full dependency graph. NetworkX turns it into answers. 40 lines.*

BLACK SWAN 4 — Apache Arrow Flight: IBM i at Memory Speed

Move Db2 data to DuckDB/pandas/Spark at Gb/s with zero serialisation overhead

```
import pyarrow as pa, pyarrow.flight as fl
import ibm_db_dbi, pandas as pd

# IBM i side: Arrow Flight server (runs in PASE)
class IBMiFlightServer(fl.FlightServerBase):
    def do_get(self, ctx, ticket):
        conn = ibm_db_dbi.connect('*LOCAL','FLIGHTUSER','***')
        df   = pd.read_sql(ticket.ticket.decode(), conn)
        return fl.RecordBatchStream(pa.Table.from_pandas(df))

# Client side (anywhere on the network)
client = fl.FlightClient('grpc://my-ibmi.cdinvest.be:8815')
reader = client.do_get(fl.Ticket(
    b"SELECT ARTNO,LISTPRICE FROM JORI.JRARTIK"))
table = reader.read_all()  # Arrow columnar, no copy
import duckdb; duckdb.connect().register('sofas', table)
# → pipe to Superset, Spark, Polars: zero conversion
```

💡 *Arrow Flight: IBM i → DuckDB/pandas/Spark at memory speed. No ODBC. No conversion. No bottleneck.*

KEY TAKEAWAYS

The Vault Is Open

All on your machine. None requiring a new purchase.

1. SQL Services + Db2: QSYS2 turns your OS into a database. FOR SYSTEM_TIME = time travel. ENCRYPT_AES + RCAC = GDPR built in.
2. Query Supervisor + Index Advisor: Db2 governance and tuning powered by data, not guesswork.
3. Cross-System SQL + ArdGate: JOIN across IBM i partitions via DRDA, or to Oracle/Postgres/MySQL via *ARDPGM. Zero ETL.
4. Authority Collection + Exit Points: end the *ALLOBJ era. STRAUTCOL + ADDEXITPGM = least privilege from facts.
5. SIEM Logger: DISPLAY_JOURNAL(..., GENERATE_SYSLOG=>'RFC5424') + syslogd (Option 33) → QRadar/Splunk. Free since 7.2.
6. Polyglot + ActiveMQ + Service Commander: MariaDB, PostgreSQL, Redis, JMS broker — all on one LPAR, managed like systemd.
7. Ghostscript + Infoprint: PDF with stapling/duplex/finishing from CL. Production-grade print runs without third-party tools.
8. MCP + Python + WOPASE: AI talks to Db2. Python talks to Excel. WOPASE distributes ILE — all natively, on the machine.

Questions?

SQL Services • Db2 • Query Supervisor • Index Advisor • Journals • System Objects • Cross-System SQL • ArdGate • Security • Authority Collection • Exit Points • Column Encryption • SIEM Logger • DB2/SSH • PDI • Open Source • MCP • Polyglot DBs • ActiveMQ • Service Commander • Python→Excel • Ghostscript • Infoprint • OmniFind • Geospatial • VROOM • Jasper • Superset • Zeebe • WOPASE • IoT • Vectors • AI • Black Swans

Koen Decorte • CD-Invest nv • Melsele, Belgium

kdecorte@cdinvest.be • cdinvest.eu • PEPPOL: PBE000833