Nice — I’ll walk you through a complete, end-to-end playbook to build the **Financial Risk & Compliance Dashboard** that achieves those resume bullets. I’ll assume you have access to a SQL Server database (or CSVs/extracts you can import) and Power BI Desktop. I’ll show the *exact* steps, SQL examples, DAX ideas, validation checks, deployment/automation notes, and how to measure the 30% / 20% improvements. Ready? Let’s go.

**Plan overview (high level)**

1. **Gather & understand data** (sources, fields, access)
2. **Ingest & stage** data into a SQL staging schema (or single DB)
3. **Clean & normalize** (dedup, unify formats, type conversions)
4. **Transform & enrich** (business logic, window functions, flags)
5. **Anomaly detection & validation** (rules + statistical checks)
6. **Build Power BI data model** (relationships, measures, columns)
7. **Design dashboard UX** (visuals, drill-through, alerts)
8. **Testing & QA** (data correctness, performance)
9. **Automation & deployment** (scheduled refresh, access controls)
10. **Measure impact & handoff** (how to compute the 30%/20%)

**1 — Gather & understand data (1–3 days)**

Goal: know what you have and who owns it.

Steps:

1. **List data sources** — typical examples:
   * General ledger / transactions table (txn\_id, date, amount, acct\_id, dept\_id, vendor, description, status, created\_by)
   * Master tables (accounts, departments, vendors, users)
   * Bank statements, payment files, expense reports
   * Audit logs (user changes, approvals)
2. **Get samples** — ask DBAs or data owners for:
   * schema DDL or table description
   * 100–1,000 sample rows per table (CSV) to inspect formats
3. **Document columns & meanings** in a small spreadsheet: name, type, example values, cardinality, null rate, owner.
4. **Clarify business rules** with audit/finance: what counts as suspicious, high-value thresholds, duplicate definition, approved statuses.

Deliverables: source inventory + sample CSVs + rules doc.

**2 — Ingest & stage (1 day)**

Goal: create a reproducible place to clean/transform.

Steps:

1. **Create a staging schema** in SQL Server (e.g., staging.transactions\_raw) and load raw extracts into it. If you can run ETL, use SSIS or SQL bulk import; otherwise use CSV import.
2. **Store raw snapshots** (don’t overwrite) — keep raw files for traceability.

SQL to create staging table (example):

CREATE SCHEMA staging;

CREATE TABLE staging.transactions\_raw (

txn\_id VARCHAR(50),

txn\_date DATETIME2,

amount DECIMAL(18,2),

account\_code VARCHAR(50),

dept\_code VARCHAR(50),

vendor\_id VARCHAR(50),

description NVARCHAR(4000),

status VARCHAR(50),

created\_by VARCHAR(100),

imported\_at DATETIME2 DEFAULT SYSUTCDATETIME()

);

Load data using BULK INSERT / OPENROWSET or your ETL.

Deliverable: raw staging tables + load logs.

**3 — Clean & normalize (2–4 days)**

Goal: make data consistent and correct schema/types.

Steps and checks:

1. **Standardize date/time**: convert all date formats to YYYY-MM-DD / DATETIME2.
2. SELECT txn\_id,
3. TRY\_CONVERT(datetime2, txn\_date, 121) AS txn\_date,
4. ...
5. FROM staging.transactions\_raw;
6. **Normalize text**: trim, uppercase codes, remove weird characters in vendor names.
7. UPDATE staging.transactions\_raw
8. SET vendor\_id = UPPER(LTRIM(RTRIM(vendor\_id)));
9. **Fix numeric types & outliers**: cast amounts, flag negative or extreme amounts for review.
10. **Remove exact duplicates** (optional keep snapshot):
11. SELECT txn\_id, COUNT(\*) AS cnt
12. FROM staging.transactions\_raw
13. GROUP BY txn\_id
14. HAVING COUNT(\*) > 1;
15. **Identify near-duplicates** (same date, amount, vendor, description fuzzy) — tag them for manual review.
16. **Null & missing data**:
    * Create a data\_quality table logging rows with missing critical fields (txn\_id, date, amount, dept).
    * For corrections, create a staging.transactions\_clean after remediation.

Example transformation to clean table:

CREATE TABLE transit.transactions\_clean AS

SELECT DISTINCT

COALESCE(NULLIF(txn\_id,''), NEWID()) AS txn\_id,

TRY\_CONVERT(datetime2, txn\_date, 121) AS txn\_date,

ABS(TRY\_CAST(amount AS DECIMAL(18,2))) AS amount,

UPPER(LTRIM(RTRIM(dept\_code))) AS dept\_code,

...

FROM staging.transactions\_raw;

Deliverable: clean staging table + data quality log.

**4 — Transform & enrich (3–5 days)**

Goal: apply business logic, compute flags, and create analytical tables.

Steps:

1. **Create dimension tables** — dim\_department, dim\_vendor, dim\_account with canonical keys and descriptions.
2. **Create a fact table** fact\_transactions with normalized foreign keys and precomputed fields.
3. **Engineered columns**: fiscal period, week, is\_high\_value, is\_cash, approval\_lag\_days, user\_risk\_score.

Example SQL for enrichment and window functions (detect duplicates and high-value):

-- Identify duplicated amounts within same account and date

WITH ranked AS (

SELECT

txn\_id, txn\_date, amount, account\_code, vendor\_id,

ROW\_NUMBER() OVER (PARTITION BY account\_code, txn\_date, amount ORDER BY txn\_id) AS rn,

COUNT(\*) OVER (PARTITION BY account\_code, txn\_date, amount) AS dup\_count

FROM transit.transactions\_clean

)

SELECT \*,

CASE WHEN dup\_count > 1 THEN 1 ELSE 0 END AS is\_exact\_duplicate,

CASE WHEN amount >= 100000 THEN 1 ELSE 0 END AS is\_high\_value

INTO transit.transaction\_flags

FROM ranked;

1. **Suspicious pattern detection** (examples):
   * Rapid repeat payments to same vendor within short window:
   * SELECT t.\*
   * FROM (
   * SELECT \*,
   * LAG(amount) OVER (PARTITION BY vendor\_id ORDER BY txn\_date) AS prev\_amount,
   * DATEDIFF(day, LAG(txn\_date) OVER (PARTITION BY vendor\_id ORDER BY txn\_date), txn\_date) AS days\_since\_prev
   * FROM transit.transactions\_clean
   * ) t
   * WHERE days\_since\_prev <= 7 AND ABS(amount - prev\_amount) < 1;
   * Vendor payment to self (vendor\_id matches payee user)
   * Round-number payments (amount % 1000 = 0) — sometimes suspicious
2. **Scoring / composite risk score** — combine flags:

ALTER TABLE transit.transaction\_flags ADD risk\_score AS (

is\_exact\_duplicate\*3 + is\_high\_value\*4 + suspicious\_pattern\_flag\*5

);

Deliverable: fact\_transactions and transaction\_flags tables with risk\_score.

**5 — Anomaly detection & validation (2–4 days)**

Goal: implement both business-rule and statistical anomaly detection and validate results.

Approach:

1. **Rule-based anomalies** (deterministic):
   * Is duplicate
   * Is high-value (>= threshold, e.g., 100k)
   * Vendor blacklisted
   * Approval lag > x days
2. **Statistical anomalies**:
   * Z-score per vendor/account: transactions that are X standard deviations from mean
   * Time-series anomaly: sudden spike in count or amount per department
   * Use simple moving average and percent-change thresholds.

SQL example for z-score by vendor:

WITH stats AS (

SELECT vendor\_id,

AVG(amount) AS mean\_amt,

STDEV(amount) AS sd\_amt

FROM transit.transactions\_clean

GROUP BY vendor\_id

)

SELECT t.\*,

(t.amount - s.mean\_amt) / NULLIF(s.sd\_amt,0) AS z\_score

FROM transit.transactions\_clean t

JOIN stats s ON t.vendor\_id = s.vendor\_id

WHERE (t.amount - s.mean\_amt) / NULLIF(s.sd\_amt,0) > 3;

1. **Combine rules into final anomaly flag**:

UPDATE transit.transaction\_flags

SET final\_anomaly\_flag = CASE

WHEN is\_exact\_duplicate = 1 THEN 1

WHEN is\_high\_value = 1 AND z\_score > 2 THEN 1

WHEN suspicious\_pattern\_flag = 1 THEN 1

ELSE 0

END;

1. **Validation**:
   * Random sample review of flagged vs non-flagged transactions by auditors.
   * Confusion matrix: true positives, false positives, false negatives (use manual labels).
   * Track precision and recall; iterate rules to balance.

Deliverable: transaction\_anomalies table + validation notebook/log.

**6 — Build Power BI data model (2–4 days)**

Goal: create optimized model for fast, secure reporting.

Steps:

1. **Import data** from SQL Server (recommended: import mode for performance unless dataset > few million rows — then use DirectQuery or composite).
   * Tables: fact\_transactions, dim\_department, dim\_vendor, dim\_account, dim\_date
2. **Create relationships** (star schema): fact -> dims by keys.
3. **Set columns types**, mark date table as Mark as Date Table.
4. **Create calculated columns & measures (DAX)**:
   * Basic KPIs:
   * TotalAmount = SUM(fact\_transactions[amount])
   * TotalAnomalies = CALCULATE(COUNTROWS(fact\_transactions), FILTER(fact\_transactions, fact\_transactions[final\_anomaly\_flag] = 1))
   * AnomalyRate = DIVIDE([TotalAnomalies], COUNTROWS(fact\_transactions))
   * High-value count:
   * HighValueCount = CALCULATE(COUNTROWS(fact\_transactions), fact\_transactions[is\_high\_value] = 1)
   * Avg approval lag:
   * AvgApprovalLag = AVERAGE(fact\_transactions[approval\_lag\_days])
   * Risk-weighted exposure:
   * RiskExposure = SUMX(fact\_transactions, fact\_transactions[amount] \* fact\_transactions[risk\_score])
5. **Optimization**:
   * Hide unnecessary columns, reduce cardinality (e.g., shorten strings).
   * Use aggregations or pre-aggregated tables for very large data.

Deliverable: Power BI .pbix model with measures.

**7 — Dashboard UX & visuals (2–4 days)**

Goal: make an intuitive, audit-focused dashboard.

Suggested pages:

1. **Executive Overview** — KPIs across 5 departments:
   * Total transactions, total amount, anomalies count, anomaly rate, high-value exposure, trend sparkline.
2. **Department Drill-down**:
   * Table + bar chart: anomalies by department, anomalies timeline, top vendors by anomaly count.
3. **Transaction Explorer**:
   * Searchable table with drill-through to transaction detail page (txn\_id, date, amount, risk flags, notes).
4. **Anomaly Investigator**:
   * Use filters: vendor, date-range, risk\_score; show contextual visuals (prior transactions, account history).
5. **Audit Logs & Evidence**:
   * Attach links to source files or show data lineage and last updated timestamps.

Power BI UX tips:

* Use conditional formatting to highlight risk (red/orange/green; user can choose colors).
* Add tooltips showing rule reasons (e.g., Flagged because: Duplicate & High-value).
* Add bookmarks for common views (e.g., “High-Risk Today”).
* Enable row-level security if needed (restrict by department).

Deliverable: finalized .pbix with 4–5 pages + instructions.

**8 — Testing & QA (2–3 days)**

Goal: ensure accuracy, performance, and stakeholder acceptance.

Tests:

1. **Data correctness**:
   * Reconcile totals between SQL and Power BI (e.g., SELECT COUNT(\*) vs Power BI counts).
   * Spot-check 50 transactions across different flags.
2. **Performance**:
   * Page load target < 5 seconds for common views. Use query diagnostics in Power BI.
3. **User Acceptance Testing**:
   * Walk audit/finance through demo; capture feedback and update rules or visuals.
4. **Edge cases**:
   * Very old or future dates, nulls, negative amounts.

Deliverable: QA report + sign-off sheet.

**9 — Automation & deployment (1–3 days)**

Goal: scheduled refreshes, access, and alerts.

Steps:

1. **Publish** to Power BI Service / tenant workspace.
2. **Schedule refresh** (daily/hourly depending on need). Use gateway if on-prem SQL Server.
3. **Set Data Alerts** for KPIs (Power BI supports alerts on tiles) and/or wire to email/Teams via Power Automate for critical anomalies.
4. **Row-Level Security (RLS)** — implement if departments should only see their own data.
5. **Documentation & runbook** — include source queries, refresh schedule, owners, runbook for failures.

Deliverable: Published report + runbook + refresh schedule + alert config.

**10 — Measure impact (how to show 30% manual effort reduction & 20% efficiency increase)**

You promised concrete metrics on your resume. Here’s how to measure and present them:

1. **Baseline measurement (before automation)**:
   * Time auditors spent on manual validation in a typical audit cycle (survey or timesheets). E.g., 40 hours per cycle.
   * Number of manual exceptions processed.
2. **After automation**:
   * Time reduced by result of automated flags & validation (e.g., down to 28 hours). Compute reduction = (40 - 28)/40 = 30%.
   * Track monthly audit cycle time or number of exceptions manually reviewed.
3. **Compliance efficiency (20% improvement)**:
   * Define metric: average time to detect & escalate a suspicious transaction, or percent of flagged true positives found earlier.
   * Baseline: e.g., average detection-to-action time = 10 days. After dashboard: 8 days -> improvement = (10-8)/10 = 20%.
4. **Record these metrics** for at least 1–2 audit cycles and capture screenshots and logs as evidence (Power BI usage metrics, reduced manual hours, audit sign-off times).

Deliverable: measurement report with before/after numbers, screenshots, and stakeholder testimonial.

**Extra: Sample SQL & DAX snippets (compact)**

**SQL: Mark suspicious by composite rule**

UPDATE transit.transaction\_flags

SET suspicious\_pattern\_flag = CASE

WHEN dup\_count > 1 THEN 1

WHEN amount >= 100000 THEN 1

WHEN vendor\_id IN (SELECT vendor\_id FROM dim\_blacklist) THEN 1

ELSE 0

END,

final\_anomaly\_flag = CASE WHEN suspicious\_pattern\_flag = 1 OR z\_score > 3 THEN 1 ELSE 0 END;

**DAX: Anomaly Rate by Department**

AnomalyRate\_Dept =

DIVIDE(

CALCULATE(COUNTROWS(fact\_transactions), fact\_transactions[final\_anomaly\_flag] = 1),

CALCULATE(COUNTROWS(fact\_transactions))

)

**Deliverables checklist (what you should hand over)**

* staging raw tables + load scripts
* transit.transactions\_clean, transaction\_flags, transaction\_anomalies tables & SQL scripts
* Power BI .pbix file (model + report)
* Published Power BI dashboard (workspace link), refresh schedule & runbook
* Data quality report + validation evidence
* Measurement report showing manual-effort reduction and compliance improvements
* Short user guide (how to use filters, how to investigate a flagged txn)

**Typical timeline summary (estimate if one person)**

* Data discovery & access: 1–3 days
* Staging & cleaning: 2–4 days
* Transform & anomaly rules: 3–5 days
* Power BI model & dashboard: 3–5 days
* Testing & deployment: 2–4 days
* Total: **~2–3 weeks** end-to-end (can be faster with existing data access and SMEs)

**Final tips & governance**

* **Start simple**: deploy a Minimum Viable Dashboard with rule-based flags, then add statistical detection.
* **Iterate with auditors** weekly; their feedback will refine rules and thresholds.
* **Log everything** — who ran what script and when; auditors love traceability.
* **Avoid too many false positives**; noisy dashboards get ignored. Tune for precision initially.
* **Keep an evidence trail**: links to the raw row (txn\_id) and source file for auditors.