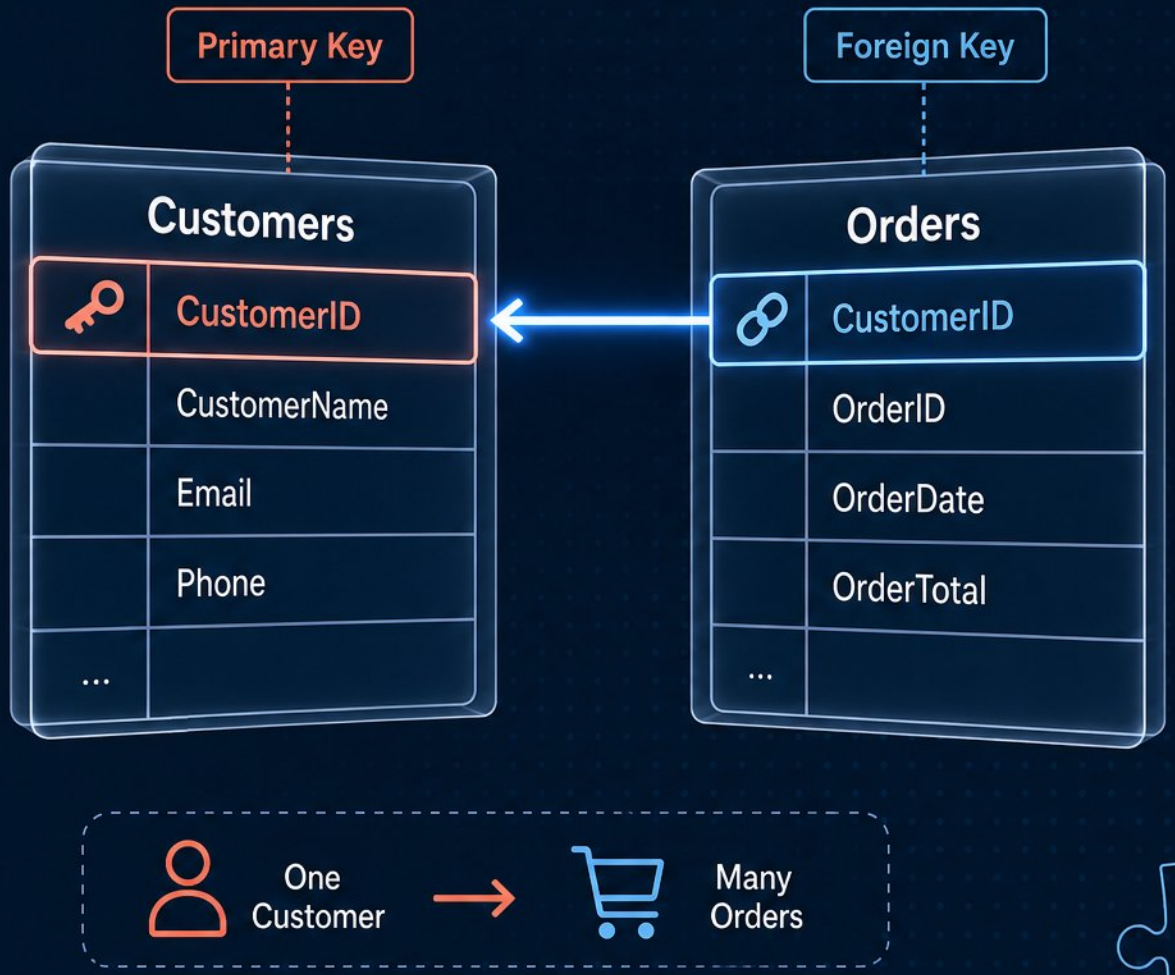
Introduction to Data Joins:

Matching Rows and Tables

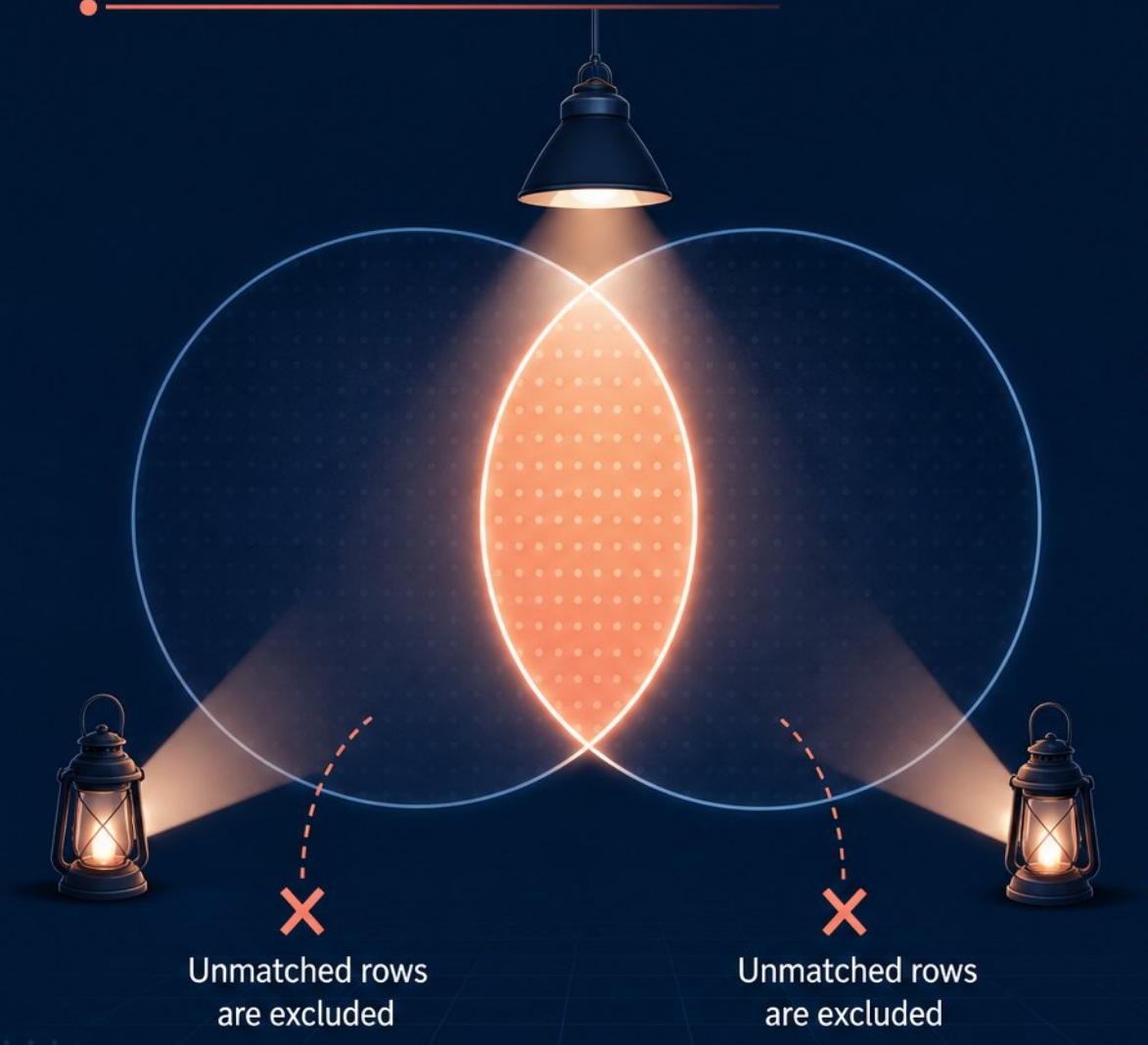
- Combine info from multiple tables for a complete business picture
- Real-world data is scattered: customer, order, and product tables
- Correct joins prevent mismatches, missing data, and silent report errors
- Roadmap: keys, inner/left joins, unmatched values, common pitfalls, practice

How Tables Relate: Keys and Relationships

- **Primary key:** uniquely identifies every row in a table, like a customer ID
- **Foreign key:** references a primary key in another table to create links
- **One-to-many:** one customer → many orders; foreign key lives on the "many" side
- **Visual:** lines connect matching columns, arrows show relationship direction
- **Pitfall:** using a non-unique key silently multiplies rows later



Inner Join: Keeping Only Matched Rows



- An INNER JOIN returns rows only when the key matches in both tables
- Unmatched rows — such as customers with no orders — are excluded completely
- Missing or invalid key values in either table cause rows to disappear silently
- Use for clean reports: "show only orders that have a valid customer"

Left Join:
Keeping All Rows from the First Table

- Preserves every row from the left table, regardless of matches
- Right table columns fill with NULL when no match exists
- Compared to inner join: unmatched left rows appear instead of vanishing
- Business use: list all customers, including those with no orders

Inner vs. Left Join: Choosing the Right Tool

- Inner join: no match means no row in the result
- Left join: no match means row stays with NULLs
- Accidental inner joins silently drop legitimate rows
- Default to left join when matches are uncertain
- Row count shrinking signals wrong join choice

Other Join Types: Right, Full Outer, and Cross Joins

- **RIGHT JOIN:** mirror of LEFT JOIN — keeps all right-table rows; swap tables and use LEFT JOIN instead
- **FULL OUTER JOIN:** keeps every row from both tables, NULLs where unmatched — ideal for data reconciliation
- **CROSS JOIN:** Cartesian product — every row paired with every other; use for size×color grids or date scaffolds
- **Rule of thumb:** LEFT JOIN ~80% of time, INNER ~15%, others ~5% — know them, but reach for them sparingly

Understanding Unmatched Values: Where NULLs Come From

NULLs signal missing matches — not errors, but clues about absent data

Causes: key typos, format mismatches, incomplete data, genuine missing links

Spot unmatched rows: filter with WHERE right_table.key IS NULL after left join

Business value: find customers who never ordered or products never sold

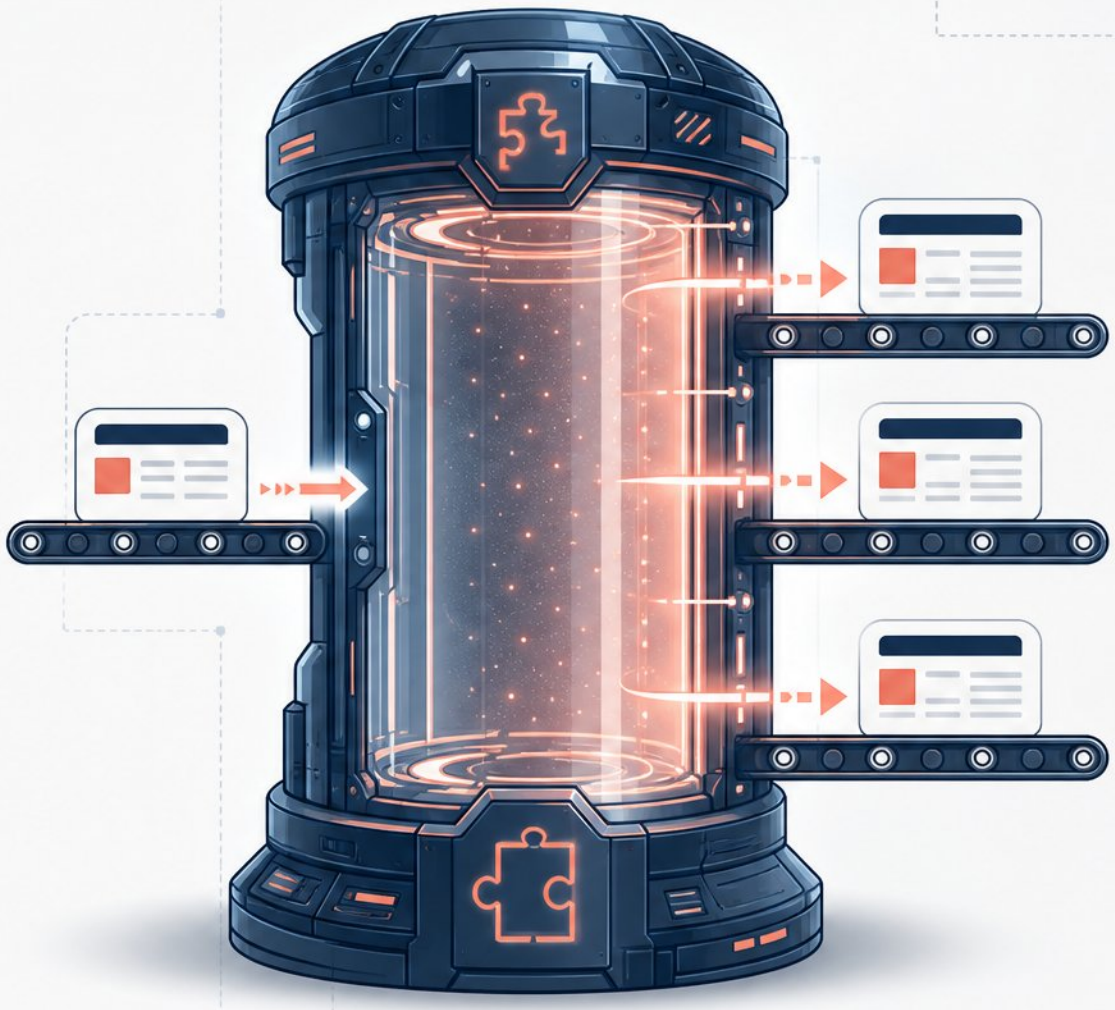
Use a non-nullable right-table key (like order_id) for reliable NULL checks

The WHERE vs. ON Trap: Accidentally Breaking Left Joins

- Filtering right table in WHERE turns left join into inner join
- WHERE o.status = 'shipped' drops rows where status is NULL
- Fix: move right-table filters into the ON clause
- Rule: right-table filters go in ON; left-table filters go in WHERE

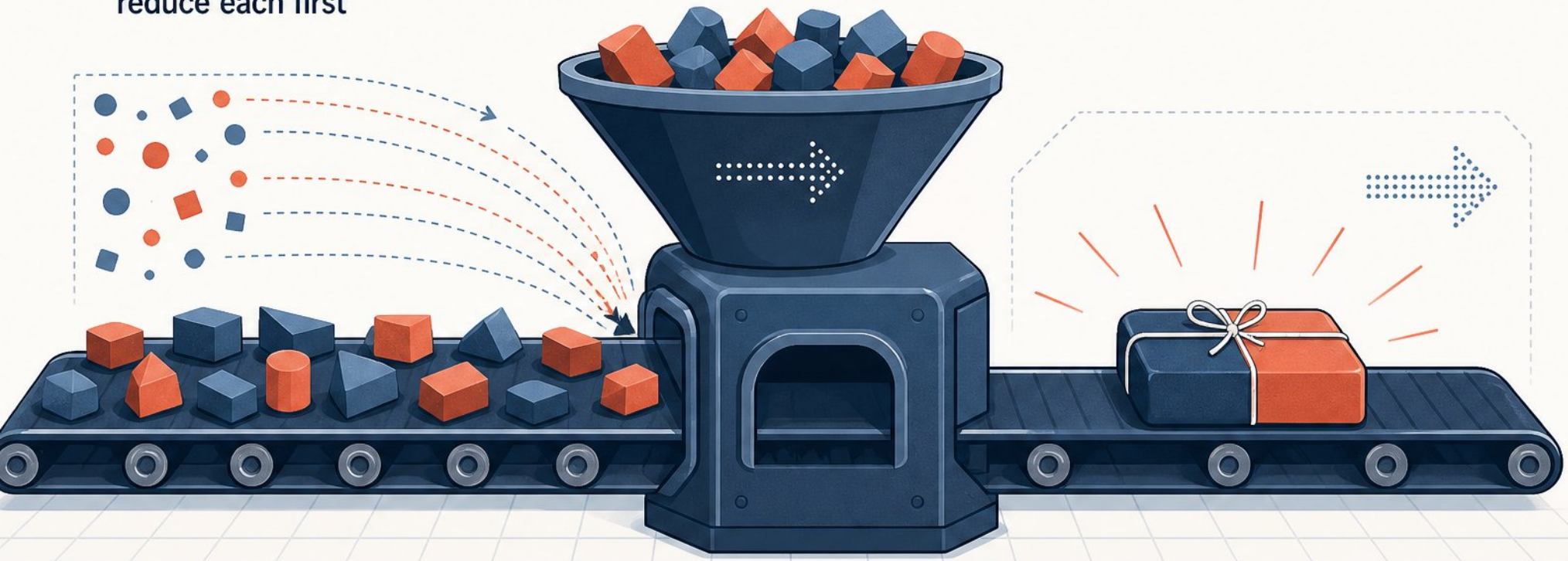
Avoiding Join Pitfalls:
Duplicate Keys and Row Multiplication

- Duplicate keys cause fan-out: one left row matches many right rows, repeating left data
- Summing order totals after joining to line items inflates revenue by the item count
- Compare COUNT(*) after the join to the left table's row count — larger means fan-out
- Always check key uniqueness (GROUP BY key, COUNT(*) > 1) before joining
- Count rows before and after every join as a quick sanity check



Fixing Duplicate Key Problems: Aggregate Before You Join

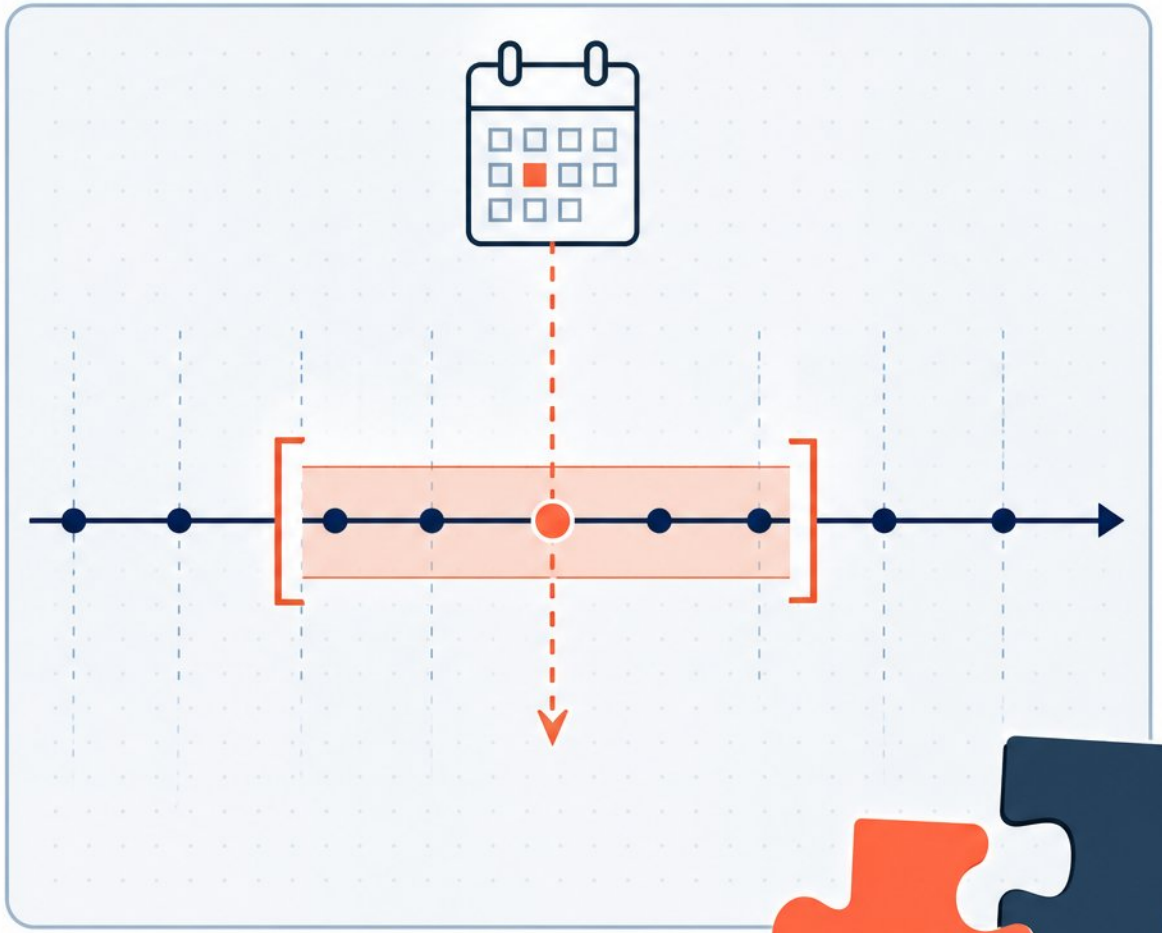
- **Match the grain: sum columns at the join's level,** not the parent's level
- **Fix 1:** sum line_total (item grain) instead of order_total (order grain)
- **Fix 2:** pre-aggregate the many-side to one row per key before joining
- **Many-to-many danger:** joining two child tables creates a cross product — reduce each first



Join Conditions Beyond Equality: Range and Non-Equi Joins

- Non-equi joins match on inequalities: >, <, >=, <=, BETWEEN, !=
- Example: Assign each sale to a price tied to its active date window

  `sale_date BETWEEN valid_from AND valid_to`
- Another example: Assign employees to salary grades based on salary ranges
- Performance tip: Pair range conditions with an equality check to avoid slow scans

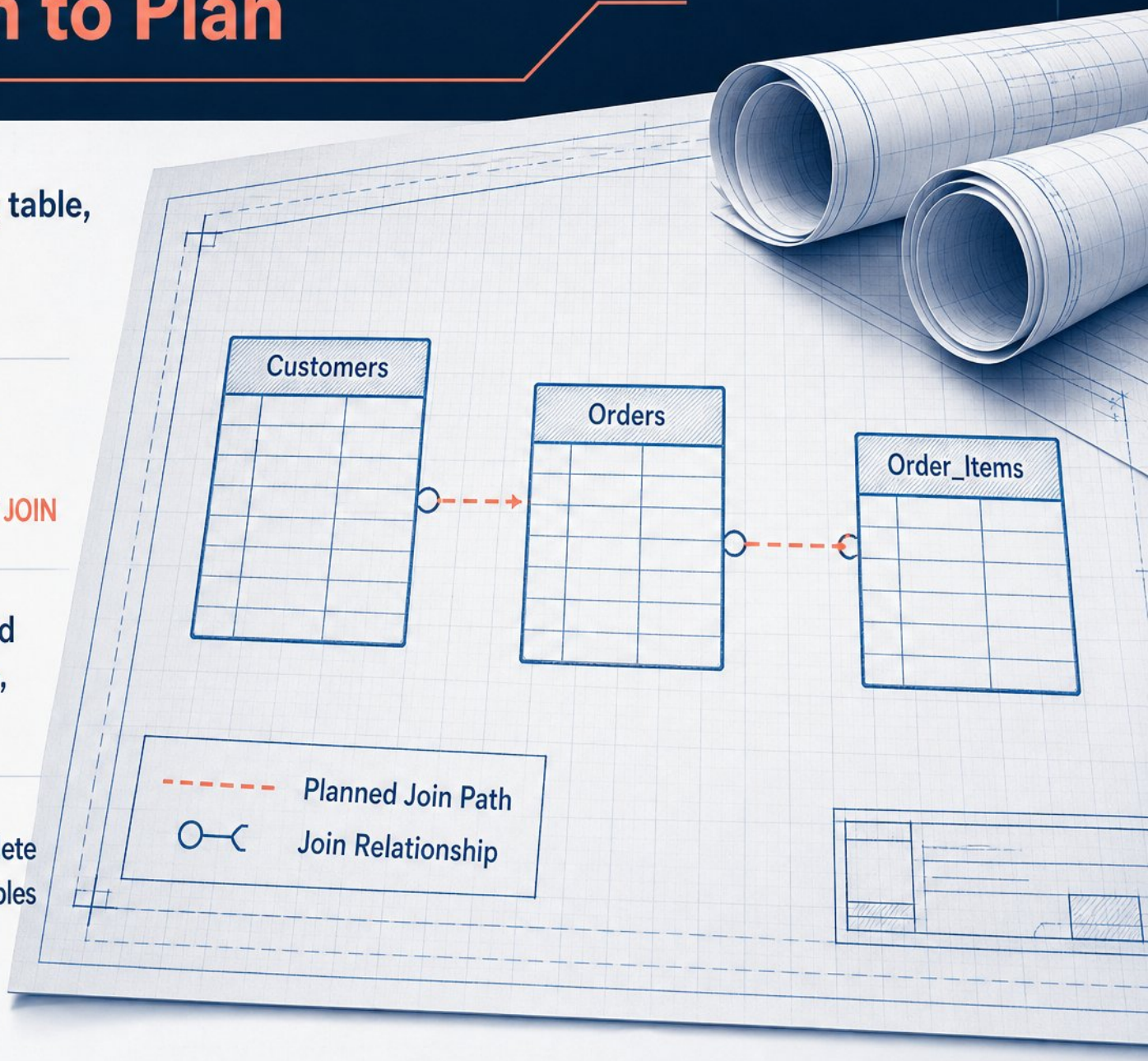


Dealing with NULLs After a Join: COALESCE and Defaults

- NULLs from unmatched joins break arithmetic and comparisons — use COALESCE to supply defaults
- COALESCE(order_total, 0) for aggregates; COALESCE(category, 'Unknown') for display labels
- The WHERE-vs-ON trap: filtering right-table columns in WHERE silently drops unmatched rows
- Distinguish "missing match" NULLs from "genuinely unknown" NULLs to avoid hiding data quality issues

Reading and Writing Join Logic: From Problem to Plan

- Identify the driving table, lookup table, and matching keys
- Translate requests: “all customers and their last order” → LEFT JOIN
- Verify plan: expected row count, NULL side, unique keys?
- **Practice:** build a complete order report across 3+ tables



Key Takeaways and Next Steps

PersonWise.

PersonWise • AI digital-human course platform

Scan the code or click anywhere to watch this course live, with voice Q&A

personwise.ai/t/094f1cb6/public