# Z2004: Worked SQL, Relational Algebra & Normalization Workbook

Database Management Systems (BS Core) · Modules M1–M10 · innocent@iitmz.ac.in

This workbook steps through five end-to-end database engineering problems spanning ER-to-relational translation, algebraic query-tree optimization, advanced window and recursive SQL, formal BCNF/3NF decomposition proofs, and transaction serializability verification.

## Problem 1: ER-to-Relational Mapping with Weak Entities and M:N Bridges

Design a relational DDL schema for a Zanzibar spice export cooperative tracking `Farmer(<u>FarmerID</u>, Name, Island)`, `Shipment(<u>ShipID</u>, ExportDate)`, and weak entity `LotItem(<u>LotNo</u>, Crop, WeightKg)` owned by `Shipment`. Each farmer contributes to multiple shipments with a recorded `SharePct`.

**Worked Step-by-Step Solution**

1. **Strong Entities:** Map `Shipment(ShipID INT PRIMARY`
2. **Weak Entity Ownership:** Lo key: `PRIMARY KEY (ShipID, L`
3. **M:N Junction Table:** Cr `SharePct <= 100))` with comp

## Problem 2: Relational Algebra Query Trees and Selection Pushdown

Given `Student(sid, sname, dept)` (10,000 rows) and `Enroll(sid, cid, grade)` (80,000 rows), optimize the naive query $\pi_{\text{sname}}\big(\sigma_{\text{dept='DSAI'} \wedge \text{grade}\geq 85}(\text{Student} \bowtie \text{Enroll})\big)$.

**Worked Step-by-Step Solution**

1. **Naive Cost:** Joining 10,000 students with 80,000 enrollment filtering.
2. **Algebraic Equivalence Pushdown:** Split conjunctive selectio $\pi_{\text{sname}}\Big(\big(\pi_{\text{sid, sname}}(\sigma_{\text{dept='DSAI'}}(\text{Student}))\big) \bowtie \big(\pi_{\text{sid}}(\sigma_{\text{grade}\geq 85}(\text{En}$
3. **Result:** Both inputs drop in cardinality and width (projecting join executes.

## Problem 3: CTE, Window Functions, and NULL-Safe Anti-Joins

Using `Orders(order_id, cust_id, order_date, amount)`, (a) compute a 3-order moving average per customer, and (b) list customers in `Customers(cust_id, name)` who have zero orders even if `Orders.cust_id` contains `NULL`s.

**Worked Step-by-Step Solution**

1. **Moving Window Query:** `SELECT cust_id, order_id, AVG(amount) OVER (PARTITION BY cust_id ORDER BY order_date, order_id ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS mov_avg FROM Orders;`
2. **NULL-Safe Anti-Join:** Never write `WHERE cust_id NOT IN (SELECT cust_id FROM Orders)`, because a single `NULL` causes every comparison to yield `UNKNOWN` and return zero rows. Use `NOT EXISTS`: `SELECT c.cust_id, c.name FROM Customers c WHERE NOT EXISTS (SELECT 1 FROM Orders o WHERE o.cust_id = c.cust_id);`

## Problem 4: BCNF vs. 3NF Decomposition and Lossless Join Proof

Let $R(A, B, C, D, E)$ satisfy $F = \{A \to B,\ BC \to D,\ D \to E\}$. Find all candidate keys, state the highest normal

form, decompose into BCNF, and verify lossless join.

**Worked Step-by-Step Solution**

1. **Candidate Keys:** Attributes $A$ and $C$ never appear on any RHS, so every s
   $(AC)^+ = \{A, C, B, D, E\} = R$ (via $A \to B$, $BC \to D$, $D \to E$). Thus $AC$ is
2. **Normal Form Violation:** $A \to B$ is a partial dependency on a proper sul
3. **BCNF Decomposition:**
   - First split on violating FD $D \to E$: $R_1(D, E)$ (BCNF) and $R_2(A, B, C, D$
   - Next split $R_2$ on violating FD $A \to B$: $R_{2a}(A, B)$ (BCNF) and $R_{2b}(A, C$
4. **Lossless Verification:** At step 1, $R_1 \cap R_2 = D$ and $D^+ \supseteq DE = R_1$. At s
   Every split satisfies the lossless-join condition, whereas preserving $BC \to D$

## Problem 5: Conflict Serializability Precedence Graph and 2PL Deadlocks

Analyze schedule $S = [R_1(X), R_2(X), W_1(X), W_2(Y), R_1(Y)]$ using a precedence graph and describe how Strict 2PL behaves on $S$.

**Worked Step-by-Step Solution**

1. **Conflict Pairs:** On $X$: $R_2(X) \prec W_1(X)$ creates edge $T_2 \to T_1$, while $R_1(X) \prec W_1(X)$ is same-transaction (ig
   On $Y$: $W_2(Y) \prec R_1(Y)$ creates edge $T_2 \to T_1$.
2. **Acyclicity Verdict:** Only edge $T_2 \to T_1$ exists, so the graph is acyclic and $S$ is conflict-serializable with eq
   serial order $T_2 \prec T_1$.
3. **Strict 2PL Execution:** Under Strict 2PL, if $T_1$ holds $S(X)$ and $T_2$ holds $S(X)$, neither can upgrade to $X($
   the other releases $S(X)$, creating a wait-for deadlock cycle ($T_1 \leftrightarrow T_2$) resolved by aborting the younger transact