Mathematical Logic

Propositional Logic · Predicate Logic · Inference · Tuple Relational Calculus

A complete, self-sufficient theory booklet built to solve every GATE CS PYQ on Mathematical Logic (2000–2026)

Simplified Education

Sunil Yadav

How this booklet is built: Every section is written to directly unlock a specific family of GATE PYQs — truth tables and equivalence laws, converse/inverse/contrapositive, custom truth-table operators, CNF/DNF, rules of inference & the resolution principle, predicate translation, quantifier negation, and the quantifier-distribution laws that decide most FOL validity questions. Worked examples throughout are drawn from and modelled on actual GATE questions.

Contents

1 Propositions, Connectives, and Truth Tables **5**
- 1.1 What is a Proposition? . . . 5
- 1.2 The Five Basic Connectives . . . 5
  - 1.2.1 Operator Precedence . . . 6
- 1.3 Well-Formed Formulas (WFFs) . . . 6

2 Tautology, Contradiction, Satisfiability, and Equivalence **7**
- 2.1 Classifying Formulas by Truth Table . . . 7
- 2.2 Logical Equivalence . . . 7
  - 2.2.1 The Standard Laws of Equivalence . . . 8
- 2.3 Two Methods to Test Equivalence / Validity . . . 8

3 Conditionals, Custom Operators, and Normal Forms **11**
- 3.1 Converse, Inverse, and Contrapositive . . . 11
- 3.2 Truth-Table–Defined (Custom) Binary Operators . . . 12
- 3.3 Conjunctive and Disjunctive Normal Form . . . 12

4 Rules of Inference and the Resolution Principle **13**
- 4.1 Arguments and Validity . . . 13
- 4.2 Standard Rules of Inference . . . 13
- 4.3 The Resolution Principle . . . 14

5 Predicate Logic: Predicates, Quantifiers, Free & Bound Variables **15**
- 5.1 Predicates and Domains . . . 15
- 5.2 The Two Quantifiers . . . 15
- 5.3 Negating Quantified Statements . . . 15

6 Translating English into Predicate Logic **17**
- 6.1 The Master Templates . . . 17
- 6.2 Nested Quantifiers — Order Matters! . . . 18

7 Validity and Equivalence of First-Order Formulas **19**
- 7.1 Quantifier Distribution Laws — the Heart of FOL-Validity MCQs . . . 19
- 7.2 Other Standard Valid FOL Schemas . . . 19
- 7.3 Method for "Is this FOL formula valid?" Questions . . . 20

8 Special Puzzle Types and Exam Strategy **21**
- 8.1 Knights-and-Knaves / Self-Referential Statements . . . 21
- 8.2 Quick-Reference Exam Checklist . . . 21

9 Tuple Relational Calculus: Predicate Logic Applied to Databases **23**
9.1 Syntax and Basic Translation 23
9.2 The Existential Join Pattern 24
9.3 The Universal Quantifier and NOT EXISTS / NOT IN 24
9.4 The Division Pattern: "For Every Row of Another Relation" 25
9.5 Translating SQL's `NOT IN` / `NOT EXISTS` / `EXISTS ...AND NOT EXISTS` 26
9.6 Safety of Expressions 27
9.7 Quick-Reference: TRC Translation Checklist 27

CHAPTER 1

Propositions, Connectives, and Truth Tables

1.1 What is a Proposition?

Definition

A **proposition** (or **statement**) is a declarative sentence that is *either true or false, but not both.* Its truth value is denoted **T** (true) or **F** (false).

Examples of propositions: "7 is a prime number" (T), "Delhi is the capital of France" (F).
Not propositions: questions ("Is it raining?"), commands ("Shut the door."), and open sentences containing a free variable ("$x > 5$") — the last of these becomes a *predicate*, covered in Chapter 4.

An **atomic proposition** is one with no logical connective inside it (p, q, r, ... are typically used as propositional variables). A **compound proposition** is built from atomic ones using connectives.

1.2 The Five Basic Connectives

Definition s

Name	Symbol	Read as
Negation	$\neg p$ (also $\overline{p}$, $\sim p$)	"not p"
Conjunction	$p \wedge q$	"p and q"
Disjunction	$p \vee q$	"p or q" (inclusive)
Conditional (Implication)	$p \to q$	"if p then q"
Biconditional	$p \leftrightarrow q$	"p if and only if q"

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \to q$	$p \leftrightarrow q$
T	**T**	**F**	**T**	**T**	**T**	**T**
T	**F**	**F**	**F**	**T**	**F**	**F**
F	**T**	**T**	**F**	**T**	**T**	**F**
F	**F**	**T**	**F**	**F**	**T**	**T**

GATE Tip

The conditional $p \to q$ is FALSE in exactly one row: $p = \mathbf{T}, q = \mathbf{F}$. This single fact resolves a huge fraction of GATE logic questions — whenever you must decide whether an implication is true, just check that its antecedent isn't true while its consequent is false. $p \to q$ is called **vacuously true** whenever p is **F**.

1.2.1 Operator Precedence

When parentheses are dropped, evaluate in this order (highest to lowest binding):

$$\neg \quad > \quad \wedge \quad > \quad \vee \quad > \quad \to \quad > \quad \leftrightarrow$$

Both $\to$ and $\leftrightarrow$ are usually taken as **right-associative**. GATE formulas are almost always fully parenthesised, but knowing this order helps you read compressed expressions and construct your own quickly.

1.3 Well-Formed Formulas (WFFs)

Definition

A **well-formed formula (WFF)** in propositional logic is defined recursively:

1. Every propositional variable ($p, q, r, \ldots$) and every constant ($\mathbf{T}, \mathbf{F}$) is a WFF.
2. If α is a WFF, so is $\neg\alpha$.
3. If α, β are WFFs, so are $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$, $(\alpha \to \beta)$, $(\alpha \leftrightarrow \beta)$.
4. Nothing else is a WFF.

This recursive definition is exactly why formulas like $F_1 : P \Rightarrow \neg P$ or $F_2 : (P \Rightarrow \neg P) \vee (\neg P \Rightarrow P)$ (GATE 2001) are valid, parseable formulas: each is built by applying the rules finitely many times, and you can always draw a unique *parse tree* for it.

CHAPTER 2

Tautology, Contradiction, Satisfiability, and Equivalence

2.1 Classifying Formulas by Truth Table

Definition

Let φ be a compound proposition over variables $p, q, r, \ldots$ Build its truth table (one row per combination of truth values). Then:

- φ is a **tautology** (or *valid*) if it is **T** in *every* row.
- φ is a **contradiction** (or *unsatisfiable*) if it is **F** in *every* row.
- φ is **satisfiable** if it is **T** in *at least one* row.
- φ is a **contingency** if it is **T** in some rows and **F** in others (satisfiable but not a tautology).

GATE Tip

Relationship chain: tautology $\Rightarrow$ satisfiable. Contradiction $\Rightarrow$ *not* satisfiable. "Satisfiable" just means "not a contradiction"; it says nothing about whether it's a tautology. GATE loves testing this exact distinction (e.g. GATE 2001 Q1.3: is F_1 unsatisfiable, is F_2 valid?).

Worked Example 2.1 — Classify a formula

Classify $F_1 : P \Rightarrow \neg P$.
Solution. Truth table: when $P = \mathbf{T}$, $F_1 = \mathbf{T} \to \mathbf{F} = \mathbf{F}$. When $P = \mathbf{F}$, $F_1 = \mathbf{F} \to \mathbf{T} = \mathbf{T}$. So F_1 is **T** in one row, **F** in the other $\Rightarrow$ F_1 **is satisfiable but not valid** (a contingency), and in particular *not* unsatisfiable. (This immediately eliminates any option claiming F_1 is unsatisfiable.)

2.2 Logical Equivalence

Definition

Two formulas α and β are **logically equivalent**, written $\alpha \equiv \beta$, if they have identical truth values in every row of their combined truth table — equivalently, $\alpha \leftrightarrow \beta$ is a tautology.

2.2.1 The Standard Laws of Equivalence

Table of Logical Equivalences

Law	Equivalence
Identity	$p \wedge \mathbf{T} \equiv p, \quad p \vee \mathbf{F} \equiv p$
Domination	$p \vee \mathbf{T} \equiv \mathbf{T}, \quad p \wedge \mathbf{F} \equiv \mathbf{F}$
Idempotent	$p \wedge p \equiv p, \quad p \vee p \equiv p$
Double negation	$\neg\neg p \equiv p$
Commutative	$p \wedge q \equiv q \wedge p, \quad p \vee q \equiv q \vee p$
Associative	$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$ (similarly for $\vee$)
Distributive	$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$
	$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$
De Morgan's	$\neg(p \wedge q) \equiv \neg p \vee \neg q$
	$\neg(p \vee q) \equiv \neg p \wedge \neg q$
Absorption	$p \vee (p \wedge q) \equiv p, \quad p \wedge (p \vee q) \equiv p$
Negation	$p \vee \neg p \equiv \mathbf{T}, \quad p \wedge \neg p \equiv \mathbf{F}$
Conditional elimination	$p \to q \equiv \neg p \vee q$
Contrapositive	$p \to q \equiv \neg q \to \neg p$
Negation of conditional	$\neg(p \to q) \equiv p \wedge \neg q$
Biconditional	$p \leftrightarrow q \equiv (p \to q) \wedge (q \to p) \equiv (p \wedge q) \vee (\neg p \wedge \neg q)$
Exportation	$(p \wedge q) \to r \equiv p \to (q \to r)$

GATE Tip

The single most useful identity on this whole exam: $p \to q \equiv \neg p \vee q$. Whenever a question buries you in nested implications, immediately rewrite every $\to$ using this rule and then apply De Morgan's / distributive laws freely. Nearly every "which formula is equivalent to / negation of" question collapses this way.

Worked Example 2.2 — Proving equivalence algebraically

Show $(p \to (q \vee r)) \equiv (\neg p \vee q \vee r)$.
Solution. $p \to (q \vee r) \equiv \neg p \vee (q \vee r)$ (conditional elimination) $\equiv \neg p \vee q \vee r$ (associativity). ■

2.3 Two Methods to Test Equivalence / Validity

1. **Truth table method:** build the full table (guaranteed correct, but 2^n rows for n variables — slow for $n \geq 4$).

2. **Algebraic method:** repeatedly apply the equivalence laws of §2.2 until both sides match, or until the formula reduces to **T** (tautology) or **F** (contradiction).

GATE Tip

On GATE, with 3–4 variables, a *partial* truth table is often the fastest approach: test only the "dangerous" rows where an implication's antecedent is true, since that's the only place $\rightarrow$ can fail.

CHAPTER 3

Conditionals, Custom Operators, and Normal Forms

3.1 Converse, Inverse, and Contrapositive

Definition

For the conditional $p \to q$:

Name	Formula
Original	$p \to q$
Converse	$q \to p$
Inverse	$\neg p \to \neg q$
Contrapositive	$\neg q \to \neg p$

Key Fact

The **contrapositive is always logically equivalent to the original** ($p \to q \equiv \neg q \to \neg p$). The **converse and inverse are equivalent to each other** ($q \to p \equiv \neg p \to \neg q$, by contrapositive applied to the converse), but neither is, in general, equivalent to the original.

Worked Example 3.1 — Converse of a natural-language conditional (GATE 2001)

Statement: "I stay only if you go."
Solution. "A only if B" translates to $A \to B$ (not $B \to A$ — this phrase is a common trap, see the pitfall box below). So here A ="I stay", B ="you go", giving stay $\to$ go. Its **converse** swaps the two sides: go $\to$ stay, i.e. **"I stay if you go."**

Common Pitfall

"A only if B" is $A \to B$, NOT $B \to A$. Students very often flip this. Test it with an example: "I will pass *only if* I study" means passing requires studying — pass $\to$ study, i.e. if I didn't study I certainly didn't pass. It does *not* mean studying guarantees passing. Contrast with **"A if B"** $= B \to A$, and **"A if and only if B"** $= A \leftrightarrow B$.

3.2 Truth-Table–Defined (Custom) Binary Operators

GATE occasionally defines a brand-new binary connective purely by its truth table (e.g. $\odot$, $\square$, $\neq$) and asks you to reason about it. Treat this exactly like any connective: **read truth values directly off the given table row by row** — do not try to guess a familiar name for it.

Worked Example 3.2 — Custom operator (GATE 2006 style)

$A \odot B$ is given by: $\mathbf{T} \odot \mathbf{T} = \mathbf{T}$, $\mathbf{T} \odot \mathbf{F} = \mathbf{T}$, $\mathbf{F} \odot \mathbf{T} = \mathbf{F}$, $\mathbf{F} \odot \mathbf{F} = \mathbf{T}$. (Notice: this is exactly the table of $A \to B$ with... check row by row — actually here it matches $A \vee \neg B$.) Which known expression equals $\neg(A \odot B)$?

Solution method. First identify $A \odot B$ in terms of standard connectives by comparing its column to standard tables: here $A \odot B \equiv (A \vee \neg B)$ [verify: TT→T∨F=T✓, TF→T∨T=T✓, FT→F∨F=F✓, FF→F∨T=T✓]. Then $\neg(A \odot B) \equiv \neg A \wedge B$ by De Morgan's.

GATE Tip

Checking commutativity/associativity of a custom operator: $\odot$ is **commutative** iff its table is symmetric under swapping the two input columns (row A, B agrees with row B, A). Testing **associativity** $(A \odot B) \odot C = A \odot (B \odot C)$ generally needs you to build an 8-row table over A, B, C and compare both sides directly — there's no shortcut, so do it systematically.

3.3 Conjunctive and Disjunctive Normal Form

Definition

A **literal** is a propositional variable or its negation (p or $\neg p$).

- A formula is in **Disjunctive Normal Form (DNF)** if it is a disjunction (OR) of one or more conjunctions (ANDs) of literals: $(l_1 \wedge l_2 \wedge \cdots) \vee (l_1' \wedge l_2' \wedge \cdots) \vee \cdots$
- A formula is in **Conjunctive Normal Form (CNF)** if it is a conjunction (AND) of one or more disjunctions (ORs) of literals: $(l_1 \vee l_2 \vee \cdots) \wedge (l_1' \vee l_2' \vee \cdots) \wedge \cdots$

Every propositional formula has an equivalent CNF and an equivalent DNF.

Algorithm to convert any formula to CNF (or DNF):

1. Eliminate $\leftrightarrow$ using $p \leftrightarrow q \equiv (p \to q) \wedge (q \to p)$.
2. Eliminate $\to$ using $p \to q \equiv \neg p \vee q$.
3. Push negations inward to literals using De Morgan's laws and double negation.
4. Apply the distributive law repeatedly: for CNF, distribute $\vee$ over $\wedge$; for DNF, distribute $\wedge$ over $\vee$.

GATE Tip

A clause (a single OR-group in a CNF) is FALSE only if every literal in it is false — so "for every truth assignment, at least half the clauses of a CNF formula evaluate to true" type statements (GATE 2007 Q1.20) should be checked by picking small concrete counter-formulas rather than trying to reason abstractly; e.g. the single-clause CNF (p) is false for exactly one assignment out of two, i.e. half — test edge cases like a 1-clause or all-negative-literal CNF to disprove universal claims.

CHAPTER 4

Rules of Inference and the Resolution Principle

4.1 Arguments and Validity

Definition

An **argument** consists of a list of premises $H_1, H_2, \ldots, H_n$ and a conclusion C. The argument is **valid** if C is true whenever *all* premises are true, i.e.

$$(H_1 \wedge H_2 \wedge \cdots \wedge H_n) \rightarrow C \quad \text{is a } \textit{tautology}.$$

To test validity of an argument (three interchangeable methods):

1. Build the truth table of $(H_1 \wedge \cdots \wedge H_n) \rightarrow C$ and check it's always **T**.

2. Assume all premises **T** and the conclusion **F**; if this forces a contradiction, the argument is valid (*indirect / proof-by-contradiction method* — usually fastest by hand).

3. Chain standard inference rules (below) from premises to conclusion.

Worked Example 4.1 — Testing validity by contradiction (GATE-style, cf. Q1.31)

I_1: Premise "If it rains, the match will not be played" ($R \rightarrow \neg M$); Premise "The match was played" (M). Conclusion: "There was no rain" ($\neg R$). Is I_1 valid?
Solution. Assume premises **T**, conclusion **F**: so $M = \mathbf{T}$ (given), and $\neg R = \mathbf{F} \Rightarrow R = \mathbf{T}$. Check first premise with $R = \mathbf{T}$: $R \rightarrow \neg M = \mathbf{T} \rightarrow \neg\mathbf{T} = \mathbf{T} \rightarrow \mathbf{F} = \mathbf{F}$. But we assumed all premises are **T**— contradiction. Hence it's impossible for premises **T** and conclusion **F** simultaneously $\Rightarrow I_1$ **is a valid inference.**

4.2 Standard Rules of Inference

Inference Rules (each is a tautology of the form Premises → Conclusion)

Rule	Premises	Conclusion
Modus Ponens	$p \to q,\ p$	q
Modus Tollens	$p \to q,\ \neg q$	$\neg p$
Hypothetical Syllogism	$p \to q,\ q \to r$	$p \to r$
Disjunctive Syllogism	$p \vee q,\ \neg p$	q
Addition	p	$p \vee q$
Simplification	$p \wedge q$	p
Conjunction	$p,\ q$	$p \wedge q$
Resolution	$p \vee q,\ \neg p \vee r$	$q \vee r$

4.3 The Resolution Principle

Definition

Resolution is the single inference rule: from clauses $(P \vee Q)$ and $(\neg Q \vee R)$, derive $(P \vee R)$ — the complementary literals $Q, \neg Q$ "cancel". It operates on clauses (disjunctions of literals) and is the backbone of automated theorem proving / logic programming, because it is the *only* rule needed once every formula is converted to CNF (**resolution refutation**: to prove $S \models C$, show $S \wedge \neg C$ is unsatisfiable by resolving clauses down to the empty clause).

Worked Example 4.2 — Applying and verifying resolution (GATE 2003, Q1.7 pattern)

From $(P \vee R)$ and $(Q \vee \neg R)$, resolution derives $(P \vee Q)$ (resolving on $R, \neg R$). Verify $(P \vee R) \wedge (Q \vee \neg R) \to (P \vee Q)$ is a tautology.
Solution. Check the only row that could break it: try to make $(P \vee Q)$ **F**, i.e. $P = \mathbf{F}, Q = \mathbf{F}$. Then $(P \vee R) = R$ and $(Q \vee \neg R) = \neg R$; both premises true means $R = \mathbf{T}$ and $\neg R = \mathbf{T} \Rightarrow R = \mathbf{F}$ — contradiction (R can't be both). So premises can never both be true while the conclusion is false $\Rightarrow$ valid. This is the standard way to verify any proposed resolution/inference statement in a GATE MCQ: try to falsify the conclusion and show the premises then can't all hold.

GATE Tip

When a GATE question gives you *four* statements about a resolution/implication rule and asks which is **FALSE**, test each with the contradiction method above rather than building a full truth table — it's far faster for 3-variable clauses.

CHAPTER 5

Predicate Logic: Predicates, Quantifiers, Free & Bound Variables

5.1 Predicates and Domains

Definition

A **predicate** $P(x)$ is a statement whose truth value depends on the variable(s) it contains; it becomes a proposition only once x is assigned a value from some **domain (universe of discourse)**, or once it is **quantified**. E.g. $P(x)$: "x is prime" is not itself true or false until x is fixed (e.g. $P(7) = \mathbf{T}$) or bound by a quantifier ($\forall x\, P(x)$, $\exists x\, P(x)$).

5.2 The Two Quantifiers

Symbol	Read as	$\forall x\, P(x)$ / $\exists x\, P(x)$ **is true when…**
$\forall x\, P(x)$	"for all x, $P(x)$"	$P(x)$ holds for *every* element of the domain
$\exists x\, P(x)$	"there exists x such that $P(x)$"	$P(x)$ holds for *at least one* element

Definition

An occurrence of a variable is **bound** if it lies within the scope of a quantifier over that variable; otherwise it is **free**. A formula with no free variables is a **sentence** (closed formula) and has a fixed truth value once the domain and predicates are fixed. In $\forall x\,[\alpha] \to (\exists y, \beta \to (\forall u, \exists v, \gamma))$, all of x, y, u, v are bound; a formula like $\beta(x)$ appearing with *no* enclosing quantifier over x has x free.

5.3 Negating Quantified Statements

Quantifier De Morgan's Laws — memorise these cold

$$\neg(\forall x\, P(x)) \equiv \exists x\, \neg P(x) \qquad \neg(\exists x\, P(x)) \equiv \forall x\, \neg P(x)$$

In words: negation flips $\forall \leftrightarrow \exists$ and pushes through onto the predicate. This is exactly analogous to De Morgan's for $\wedge/\vee$ (in fact $\forall$ behaves like an infinite $\wedge$ and $\exists$ like an infinite $\vee$ over the domain).

Worked Example 5.1 — Chained negation (GATE 2009, Q1.28 pattern)

Which of $\neg\forall x(P(x))$, $\neg\exists x(P(x))$, $\neg\exists x(\neg P(x))$, $\exists x(\neg P(x))$ are equivalent?
Solution. Apply the law to each: I. $\neg\forall x P(x) \equiv \exists x \neg P(x)$. II. $\neg\exists x P(x) \equiv \forall x \neg P(x)$. III. $\neg\exists x(\neg P(x)) \equiv \forall x \neg\neg P(x) \equiv \forall x\, P(x)$. IV. $\exists x(\neg P(x))$ — already in the target form, matches statement I's result. So **I and IV are equivalent** (both $\equiv \exists x\, \neg P(x)$); II and III are a different pair (both $\equiv$ complementary things) — always reduce every option to the same canonical form before comparing.

GATE Tip

"Not all X are Y" $\equiv \exists x(X(x) \wedge \neg Y(x))$, NOT $\forall x(X(x) \to \neg Y(x))$. Work it out via negation: "All X are Y" is $\forall x(X(x) \to Y(x))$; negating gives $\exists x \neg(X(x) \to Y(x)) \equiv \exists x(X(x) \wedge \neg Y(x))$ (using $\neg(p \to q) \equiv p \wedge \neg q$ from §2.2). This exact pattern is GATE's favourite predicate-translation trap ("Not all that glitters is gold", Q1.35; "Not every graph is connected", Q1.19).

CHAPTER 6

Translating English into Predicate Logic

6.1 The Master Templates

Universal and Existential Statement Templates

English pattern	Predicate logic form
"All A are B"	$\forall x\,(A(x) \to B(x))$
"Some A are B" / "There is an A that is B"	$\exists x\,(A(x) \wedge B(x))$
"No A is B"	$\forall x\,(A(x) \to \neg B(x)) \quad \equiv \quad \neg\exists x(A(x) \wedge B(x))$
"Not all A are B"	$\exists x\,(A(x) \wedge \neg B(x))$

Common Pitfall

The single most common error: using $\wedge$ where $\to$ belongs (or vice-versa).

- $\forall x\,(A(x) \wedge B(x))$ says *every single element of the whole domain* is both A and B — almost never what "all A are B" means (there could be non-A elements in the domain!). **Use $\to$ with $\forall$.**
- $\exists x\,(A(x) \to B(x))$ is nearly always *trivially true* (satisfied by any x that is not A, since $F \to$anything is T) and essentially never expresses "some A is B". **Use $\wedge$ with $\exists$.**

Rule of thumb: $\forall$ pairs with $\to$; $\exists$ pairs with $\wedge$. This resolves GATE 1.8, 1.16, 1.22, 1.26, 1.58 immediately.

Worked Example 6.1 — Multi-clause translation (GATE 2004, Q1.8)

"Some boys in the class are taller than all the girls." Given $\text{taller}(x, y)$.
Solution. Break it into pieces: "some boy x" $\to \exists x(\text{boy}(x) \wedge \ldots)$ [$\exists$ pairs with $\wedge$]. "x is taller than all the girls" $\to \forall y(\text{girl}(y) \to \text{taller}(x, y))$ [$\forall$ pairs with $\to$]. Combine:

$$\exists x\,\big(\text{boy}(x) \wedge \forall y\,(\text{girl}(y) \to \text{taller}(x, y))\big).$$

Worked Example 6.2 — "Only if", "unless", "none" (GATE-style, cf. Q1.33, Q1.39)

Translate: (a) "None of my friends are perfect." (b) "Not all rainy days are cold."
Solution. (a) "None of my friends are P" is the "No A is B" template: $\forall x(\text{friend}(x) \to \neg\,\text{perfect}(x))$. (b) "Not all A are B" template directly: $\exists d\,(\text{Rainy}(d) \wedge \neg\text{Cold}(d))$.

GATE Tip

"Unless" behaves like "if not": "Y unless Z" $\equiv \neg Z \to Y \equiv Z \vee Y$. So "If X then Y unless Z" (GATE 2002, Q1.4) reads as "(X and not-Z) implies Y", i.e. $(X \wedge \neg Z) \to Y$.

6.2 Nested Quantifiers — Order Matters!

Which orders commute?

$$\forall x\, \forall y\, P(x,y) \equiv \forall y\, \forall x\, P(x,y) \qquad \exists x\, \exists y\, P(x,y) \equiv \exists y\, \exists x\, P(x,y)$$

Same-type quantifiers always commute. But:

$$\exists x\, \forall y\, P(x,y) \implies \forall y\, \exists x\, P(x,y) \qquad \text{(one-directional implication only, NOT equivalence!)}$$

GATE Tip

Classic illustration: let $P(x,y)$ = "x loves y", domain = people. $\exists x\, \forall y\, P(x,y)$ says "someone loves everyone"; $\forall y\, \exists x\, P(x,y)$ says "everyone is loved by someone (possibly different lovers for different people)". The first *implies* the second but is a much stronger claim — so $\exists\forall$ and $\forall\exists$ are never interchangeable. GATE tests this directly with $M(x,y)$ ="x knows y" (Q1.60) and $F(x,y,t)$ ="x can fool y at time t" (Q1.29): read the quantifier order left-to-right, one variable's binding at a time, exactly as in this example.

Worked Example 6.3 — Reading a nested-quantifier sentence aloud (GATE 2010, Q1.29 pattern)

$F(x,y,t)$: "person x can fool person y at time t". Interpret $\forall x\, \exists y\, \exists t\, (\neg F(x,y,t))$.
Solution. Read strictly left to right: "For every x [everyone], there exists a y and a t [some person, some time] such that x cannot fool y at t." In plain English: **"everyone fails to fool somebody at some time"** = "no one can fool everyone all the time."

CHAPTER 7

Validity and Equivalence of First-Order Formulas

7.1 Quantifier Distribution Laws — the Heart of FOL-Validity MCQs

A huge share of GATE's "which of the following FOL formulas is valid" questions (Q1.5, 1.13, 1.14, 1.21, 1.23, 1.24, 1.34, 1.40's cousin, 1.47, 1.48, 1.49, 1.53, 1.56, 1.59) reduce to knowing exactly which quantifier-connective combinations distribute as a two-way equivalence and which give only a one-way implication.

The Distribution Table — memorise this

Statement	Status
$\forall x(P(x) \wedge Q(x)) \equiv \forall x P(x) \wedge \forall x Q(x)$	**Equivalence (both directions valid)**
$\exists x(P(x) \vee Q(x)) \equiv \exists x P(x) \vee \exists x Q(x)$	**Equivalence (both directions valid)**
$\forall x P(x) \vee \forall x Q(x) \;\Rightarrow\; \forall x(P(x) \vee Q(x))$	one-way valid; converse FAILS
$\exists x(P(x) \wedge Q(x)) \;\Rightarrow\; \exists x P(x) \wedge \exists x Q(x)$	one-way valid; converse FAILS

GATE Tip

Memory device: $\forall$ "likes" $\wedge$ (distributes freely both ways), $\exists$ "likes" $\vee$ (distributes freely both ways). The mixed pairs ($\forall$ with $\vee$, or $\exists$ with $\wedge$) only go one direction. **Counterexample to keep in your pocket** for the failing direction: domain $= \{1, 2\}$, $P(x)$: "x is even", $Q(x)$: "x is odd". Then $\forall x(P(x) \vee Q(x))$ is **T** (every number is even or odd) but $\forall x P(x) \vee \forall x Q(x)$ is **F** (not all are even, and not all are odd) — this single counterexample instantly disproves the converse of the one-way law and is exactly the kind of check GATE options are built to trip you on.

7.2 Other Standard Valid FOL Schemas

Additional valid equivalences/implications

$$\forall x\,(P(x) \to Q) \equiv (\exists x\,P(x)) \to Q \qquad (Q \text{ has no free } x)$$
$$\exists x\,(P(x) \to Q) \equiv (\forall x\,P(x)) \to Q \qquad (Q \text{ has no free } x\text{; note the quantifier flips!})$$
$$\forall x\,(Q \to P(x)) \equiv Q \to \forall x\,P(x)$$
$$\exists x\,P(x) \vee \exists x\,Q(x) \equiv \exists x\,(P(x) \vee Q(x)) \qquad \text{(restated for emphasis)}$$

Also: renaming a bound variable never changes meaning ($\forall x\,P(x) \equiv \forall z\,P(z)$) — this is why many GATE options differ only in variable names and are secretly identical.

7.3 Method for "Is this FOL formula valid?" Questions

1. **Try to build a counterexample first.** Pick the smallest possible domain (often just $\{1,2\}$ works) and try to assign P, Q so the formula is false. If you succeed, it's **not valid**.
2. If a small domain doesn't break it, check whether the formula matches one of the known valid schema patterns above (distribution, quantifier-flip on implication, vacuous quantification).
3. For an *argument's* validity (premises $\vdash$ conclusion in FOL), the same "assume premises true, conclusion false, derive contradiction" technique from Chapter 4 works — just substitute concrete elements from the domain as needed.

Worked Example 7.1 — Disproving a converse (modelled on GATE 2020, Q1.53)

Is $\forall x(p(x) \vee W) \equiv \forall x\,p(x) \vee W$ valid, where W has no free occurrence of x?
Solution. This matches the exportation-style law $\forall x(Q \vee P(x)) \equiv Q \vee \forall x P(x)$ derivable from $\neg(\ldots)$ manipulations *only when W doesn't depend on x* — but let's verify directly rather than assume: since W is constant w.r.t. x, either W is **T** (then both sides are trivially **T**) or W is **F** (then $\forall x(p(x) \vee \mathbf{F}) \equiv \forall x\,p(x)$, matching the RHS $\forall x\,p(x) \vee \mathbf{F}$). Both cases match $\Rightarrow$ **this equivalence IS valid** (unlike the pure $\forall x P(x) \vee \forall x Q(x)$) case in §7.1's table, because here W is quantifier-free/constant, not another quantified predicate — always check first whether the "second half" is itself quantified over x or a constant with respect to x; that distinction is exactly what separates valid from invalid options in this family of questions).

Worked Example 7.2 — Checking Barbara-style syllogism validity (GATE 2005, Q1.14 pattern)

Is $(\forall x(P(x) \vee Q(x))) \Rightarrow ((\forall x P(x)) \vee (\forall x Q(x)))$ always true?
Solution. Use the pocket counterexample from the Tip box in §7.1: domain $\{1,2\}$, $P(x)$ ="even", $Q(x)$ ="odd". LHS is **T**, RHS is **F**. So the implication **fails** — this direction is NOT valid (matches the "one-way, and only the other way" row of the distribution table).

CHAPTER 8

Special Puzzle Types and Exam Strategy

8.1 Knights-and-Knaves / Self-Referential Statements

GATE occasionally embeds propositional logic inside a small verbal puzzle (truth-tellers/liars, self-referential coin-toss statements like Q1.44). **Strategy:** introduce a proposition for "person is truth-teller" (T) and for the actual fact in question (H = "result is heads"), express the spoken sentence as a biconditional or conditional linking T and H, then case-split on $T = \mathbf{T}$ vs $T = \mathbf{F}$.

Worked Example 8.1 — Self-referential statement

A person of unknown type (Type 1 always truthful, Type 2 always lies) tosses a coin and says: "The result of the toss is head if and only if I am telling the truth."
Solution. Let T: "person is telling the truth now" (so $T = \mathbf{T}$ for Type 1, and for Type 2 person's utterance to be a lie we need $T = \mathbf{F}$ meaning the biconditional statement itself is false). Let H: "result is heads". The spoken content is the claim $H \leftrightarrow T$. **Case Type 1** (always truthful, so $T = \mathbf{T}$ and the claim $H \leftrightarrow T$ must actually hold): $H \leftrightarrow \mathbf{T}$ forces $H = \mathbf{T}$. **Case Type 2** (always lies, so the claim $H \leftrightarrow T$ is actually false while $T = \mathbf{F}$ since they're lying): $H \leftrightarrow \mathbf{F}$ is false $\Rightarrow H \leftrightarrow \mathbf{F}$ false means $H \neq \mathbf{F}$, i.e. $H = \mathbf{T}$. **Both cases force $H = \mathbf{T}$ $\Rightarrow$ the result is definitely heads**, regardless of the person's type. This "both branches agree" pattern is exactly what these puzzles test for.

8.2 Quick-Reference Exam Checklist

Before you answer any Mathematical Logic MCQ, ask:

1. **Propositional or predicate logic?** If quantifiers ($\forall, \exists$) appear, jump to Ch. 5–7 tools; otherwise Ch. 1–4.
2. **"Equivalent to" or "negation of"?** Rewrite every $\rightarrow$ as $\neg p \vee q$ first (§2.2), then push negations via De Morgan's.
3. **"Valid" / "tautology" / "satisfiable"?** Recall these are three different properties (§2.1) — don't conflate them.
4. **Nested quantifiers?** Read strictly left-to-right (§6.2); never assume $\exists\forall$ and $\forall\exists$ swap freely.
5. **Distribution of $\forall/\exists$ over $\wedge/\vee$?** Use the table in §7.1 — and keep the domain-$\{1, 2\}$ counterexample ready to test the "suspicious" direction.
6. **"All/some/none/not all A are B"?** Use the templates of §6.1: $\forall$ pairs with $\rightarrow$, $\exists$ pairs with $\wedge$.

7. **Stuck?** Fall back to a truth table (propositional) or a 2-element domain (predicate) — it is slower but always correct, and multiple-choice format means you only need to eliminate 3 wrong options.

CHAPTER 9

Tuple Relational Calculus: Predicate Logic Applied to Databases

Tuple Relational Calculus (TRC) is nothing more than the predicate logic of Chapters 5–7, restricted to a database setting: tuple variables range over the rows of a relation instead of an abstract domain, and $\forall/\exists$ behave *exactly* as before. Everything you already know about quantifier translation, negation, and distribution applies directly — this chapter is about the handful of extra conventions and query patterns that are specific to databases.

9.1 Syntax and Basic Translation

Definition

A TRC query has the form

$$\{\, t.A_1, \ldots, t.A_k \mid \varphi(t) \,\}$$

where t is a **tuple variable**, $t.A_i$ denotes attribute A_i of t, and $\varphi(t)$ is a formula built from:

- **Range conditions** $t \in R$ ("t ranges over relation R"),
- **Comparisons** between attributes/constants ($=, \neq, <, >, \ldots$),
- **Connectives** $\wedge, \vee, \neg$,
- **Quantifiers** $\exists t\,(\ldots)$, $\forall t\,(\ldots)$, each tuple variable ranging over some named relation.

The result is the set of all tuples $\langle t.A_1, \ldots, t.A_k \rangle$ for which $\varphi(t)$ evaluates to true.

Worked Example 9.1 — Simple selection (cf. Q1)

"Names of all CSE students" from `STUDENT(sid,sname,dept,year)`:

$$\{\, s.sname \mid s \in STUDENT \wedge s.dept = \text{'CSE'} \,\}$$

This is just $\{x \mid P(x)\}$ set-builder notation from Chapter 5 with $P(s) \equiv (s \in STUDENT) \wedge (s.dept = \text{'CSE'})$ — no new idea, only new vocabulary ($s \in STUDENT$ plays the role of a domain restriction).

9.2 The Existential Join Pattern

The single most common TRC idiom

"t is related to *some* row of another relation R_2" is written

$$\exists u \in R_2 \left(\text{join-condition}(t, u) \wedge \text{extra-condition}(u)\right).$$

Every "has at least one", "enrolled in", "teaches", "there exists a course such that" phrase becomes exactly this existential-with-join shape — match tuple variables across relations via a shared key inside the parentheses, exactly as $\exists$ pairs with $\wedge$ in §6.1.

Worked Example 9.2 — Existential join (cf. Q2, Q19)

"Names of students who have enrolled in DBMS (`cid`=101)":

$$\{s.sname \mid s \in STUDENT \wedge \exists e \in ENROLL\,(e.sid = s.sid \wedge e.cid = 101)\}.$$

Chaining across three relations ("enrolled in a course taught by Dr. Rao", Q19): walk the join path $STUDENT \to ENROLL \to TEACHES \to INSTRUCTOR$ one $\exists$ per hop, threading the shared key at each step:

$$\{s.sname \mid s \in STUDENT \wedge \exists e \in ENROLL\, \exists t \in TEACHES\, \exists i \in INSTRUCTOR$$

$$(e.sid = s.sid \wedge e.cid = t.cid \wedge t.tid = i.tid \wedge i.tname = \text{'Dr. Rao'})\}.$$

Method: draw the join path first (which relations connect to which via which key), then write one $\exists$ per relation on the path, all inside a single $\wedge$-chain.

9.3 The Universal Quantifier and NOT EXISTS / NOT IN

This is where TRC questions get hard, and it is *exactly* the $\forall$– $\to$ vs. $\neg\exists(\wedge)$ material from Chapters 5–6, applied under database framing.

Three equivalent ways to say "every matching row satisfies X"

$$\forall e \in ENROLL\,(e.sid = s.sid \Rightarrow X(e)) \quad \equiv \quad \neg\exists e \in ENROLL\,(e.sid = s.sid \wedge \neg X(e))$$

This is exactly $\neg(p \wedge \neg q) \equiv (p \to q)$ from §2.2, applied with $p =$ "$e.sid = s.sid$" and $q = X(e)$. Both forms appear constantly in GATE/exam TRC questions, and **you must be able to convert between them instantly.**

Worked Example 9.3 — "Only A grades" and the vacuous-truth trap (cf. Q4, Q13, Q17)

$$Q = \{s.sname \mid s \in STUDENT \wedge \forall e \in ENROLL\,(e.sid = s.sid \Rightarrow e.grade = \text{'A'})\}.$$

Read this exactly as §2.2's conditional table tells you to: the inner implication is *vacuously true* for every ENROLL tuple that does *not* belong to student s (antecedent **F** $\Rightarrow$ implication **T** regardless of the consequent). So the $\forall$ only ever "bites" on rows where $e.sid = s.sid$; it silently says nothing at all about a student with zero ENROLL rows.

- **A student with no enrolment tuples trivially satisfies this predicate** — the universal quantifier over an empty set of "relevant" tuples is vacuously true (same fact as "$\forall x \in \varnothing$, $P(x)$ is always true"). This is the #1 TRC pitfall (see Q13's answer key: "Universal quantification

over an empty set is true").

- A student with grades {A, B} **fails** — the B-grade tuple makes the antecedent **T** and consequent **F**, so that one implication is **F**, making the whole $\forall$ **F**.

On the sample data, only Charu and Esha have *exclusively* A grades among students who have enrolments, giving $|Q| = 2$ (Q17's answer). Always separately check: (a) does the student have any rows at all? (b) do all their rows satisfy X? Both determine membership, but only (b) is what the formula literally encodes — a student can vacuously pass (a) = "no" and still satisfy the formula.

Common Pitfall

Never write $\forall e \in ENROLL\,(e.sid = s.sid \wedge X(e))$ using $\wedge$ instead of $\Rightarrow$. This is the identical $\forall/\wedge$ vs. $\forall/\rightarrow$ trap from §6.1, now dressed up in SQL syntax: with $\wedge$, the formula demands *every single row in the entire ENROLL relation* (not just this student's rows) satisfy both conditions — almost always false for any real table. $\forall$ **always pairs with $\Rightarrow$ in TRC, exactly as in pure predicate logic.**

9.4 The Division Pattern: "For Every Row of Another Relation"

Definition

The **division-style query** — "find X related to *every* row of relation R_2 (possibly filtered)" — nests a $\forall$ (over R_2) around an $\exists$ (over the join relation):

$$\forall c \in R_2\,\big(\text{filter}(c) \Rightarrow \exists e \in \text{JoinRel}\,(\text{match}(t,e) \wedge e.\text{key} = c.\text{key})\big).$$

Worked Example 9.4 — "Enrolled in every CSE course" (cf. Q5, Q18, Q20)

$$Q = \{s.sname \mid s \in STUDENT \wedge \forall c \in COURSE\,\big(c.dept = \text{'CSE'} \Rightarrow \exists e \in ENROLL\,(e.sid = s.sid \wedge e.cid = c.cid$$

Read it in three layers, outside-in: (1) for *every* course c in the whole COURSE relation, (2) *if* that course happens to be CSE, (3) *then* there must exist a matching enrolment by this student. Courses from other departments trivially satisfy the implication (antecedent **F**) and impose no constraint — exactly the vacuous-truth idea from §9.3, now nested one level deeper. On the sample data the CSE courses are 101, 102, 103; only Amit and Dev have all three $\Rightarrow |Q| = 2$ (matches Q18's answer).

Q20's harder variant ("enrolled in every course that has a prerequisite") nests an $\exists$ *inside the antecedent too*: the filter itself is existential ("c has *some* prerequisite"), while the consequent is the same existential-enrolment check:

$$\forall c \in COURSE\,\Big(\big(\exists p \in PREREQ\,(p.cid = c.cid)\big) \Rightarrow \big(\exists e \in ENROLL\,(e.sid = s.sid \wedge e.cid = c.cid)\big)\Big).$$

General template: $\forall$(filter $\Rightarrow$ match), and the filter itself may be any earlier-chapter formula, including its own quantifiers.

GATE Tip

Spotting a "redundant" conjunct (cf. Q9). If a query already contains $\forall c \in COURSE\,(c.dept =$ 'CSE' $\Rightarrow \exists e \ldots)$ *and* the same query separately asserts $\exists e \in ENROLL\,(e.sid = s.sid)$ ("has

at least one enrolment"), check whether the second conjunct adds anything: if the domain is guaranteed to contain at least one CSE course, the $\forall$-clause alone already forces at least one matching enrolment to exist, making the plain existential redundant. **Always ask: does the $\forall$ clause already imply the $\exists$ clause, given what's guaranteed to exist in the referenced relation?**

9.5 Translating SQL's `NOT IN` / `NOT EXISTS` / `EXISTS ...AND NOT EXISTS`

SQL $\leftrightarrow$ TRC dictionary

SQL pattern	TRC form
`WHERE x IN (SELECT ...)`	$\exists$ (matching tuple in the subquery's relation)
`WHERE x NOT IN (SELECT ...)`	$\neg\exists$ (no matching tuple)
`WHERE EXISTS (...)`	$\exists$, same condition
`WHERE NOT EXISTS (...)`	$\neg\exists$, same condition
`EXISTS(A) AND NOT EXISTS(B)`	$\exists(\ldots A\ldots) \wedge \neg\exists(\ldots B\ldots)$ — **two independent tuple variables**, one per subquery

Worked Example 9.5 — `NOT IN` and its universally-quantified twin (cf. Q3, Q11)

`WHERE s.sid NOT IN (SELECT e.sid FROM ENROLL e WHERE e.cid=101)` translates directly as

$$\neg\exists e \in ENROLL\,(e.sid = s.sid \wedge e.cid = 101).$$

By De Morgan's for quantifiers (§5.3) this is also $\forall e \in ENROLL\,\neg(e.sid = s.sid \wedge e.cid = 101)$, and by $\neg(p \wedge q) \equiv \neg p \vee \neg q$ (§2.2) that becomes

$$\forall e \in ENROLL\,(e.sid \neq s.sid \vee e.cid \neq 101).$$

All three forms are equivalent — **this is precisely why GATE-style TRC questions give "both A and C" as an answer (Q3): they are the same formula reached via two different equivalence-law routes (quantifier negation, then De Morgan's).** If you ever need to double check two TRC expressions are equivalent, push one all the way to $\neg\exists(\ldots\wedge\ldots)$ canonical form and compare.

Worked Example 9.6 — Two independent existentials (cf. Q8)

`EXISTS(A-grade row) AND NOT EXISTS(B-grade row)` needs **two separate tuple variables** (never reuse one variable name across both, since they range over possibly-different tuples):

$$\{s.sname \mid s \in STUDENT \wedge \exists e_1 \in ENROLL\,(e_1.sid = s.sid \wedge e_1.grade = \text{'A'})$$

$$\wedge\, \neg\exists e_2 \in ENROLL\,(e_2.sid = s.sid \wedge e_2.grade = \text{'B'})\}.$$

Common Pitfall

A single tuple variable cannot hold two contradictory attribute values at once. An expression like $\exists e \in ENROLL\,(e.sid = s.sid \wedge e.grade = \text{'A'} \wedge e.grade = \text{'B'})$ is a *always-false* trap: one row

cannot simultaneously have grade A and grade B, so this returns the empty set regardless of the data — it does *not* correctly express "has an A and a B" (that needs two variables, as in Ex. 9.6).

9.6 Safety of Expressions

Definition

A TRC expression is **safe** if it returns only a finite set of tuples drawn from the relations mentioned. An **unsafe** expression can (in principle) produce an infinite result — e.g. $\{t \mid \neg(t \in STUDENT)\}$ is unsafe, since it asks for every tuple in the (infinite) universe of possible tuples *not* in STUDENT. Practical query languages restrict TRC to safe expressions only; every relation-bounded expression in this chapter ($t \in R$ for a named finite relation R, with quantifiers themselves ranging over named relations) is automatically safe.

GATE Tip

Quick safety check: every free variable in the result and every quantified variable must be explicitly restricted to range over a *named, finite* relation ($t \in R$) — never left ranging over "everything." If a $\forall$ or $\exists$ in a question ranges over an unbounded/unspecified domain rather than a stated relation, flag the expression as potentially unsafe.

9.7 Quick-Reference: TRC Translation Checklist

Before answering any TRC question, ask:

1. **"At least one / some / has a"** $\rightarrow \exists$, joined with $\wedge$ (§9.2).
2. **"All / every / only"** $\rightarrow \forall$, joined with $\Rightarrow$, **never** $\wedge$ (§9.3, Pitfall box).
3. **Does the $\forall$ clause silently pass on rows outside the subject's own key?** Check vacuous truth explicitly (Ex. 9.3) — this decides whether zero-row entities are included.
4. **"For every row of another relation"** $\rightarrow$ nested $\forall(\ldots \Rightarrow \exists \ldots)$, the division pattern (§9.4).
5. `NOT IN` / `NOT EXISTS` $\rightarrow \neg\exists$; freely convert to $\forall(\neg \cdots)$ or push negation through $\wedge$ via De Morgan's if an option is phrased that way (Ex. 9.5).
6. **Two conditions on the same joined relation that could conflict (e.g. two different grades)?** Use two independent tuple variables, not one (Ex. 9.6, Pitfall box).
7. **Chained joins across 3+ relations?** Draw the join path first, then write one $\exists/\forall$ per hop, all key-conditions inside one $\wedge$-chain (Ex. 9.2).