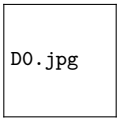
Tuple Relational Calculus

Z2004: Database Management Systems — L13

Dr. Innocent Nyalala



IIT Madras Zanzibar

30 March 2026

Lecture Outline

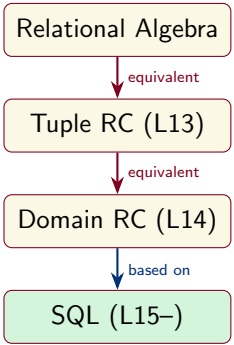
From Algebra to Calculus

Recall: Two Query Languages

- **Relational Algebra** — procedural: specifies *how* to obtain the result (sequence of operations)
- **Relational Calculus** — declarative: specifies *what* the result should contain, not how to compute it

Two Flavours of Calculus

- **Tuple RC (TRC)** — variables range over *tuples*
- **Domain RC (DRC)** — variables range over *column values*



TRC Syntax: The Notation

General Form

$$\{ t \mid P(t) \}$$

- t is a **tuple variable** — it ranges over all tuples of some relation
- $P(t)$ is a **predicate** (formula) that must be *true* for a tuple to appear in the result
- Result = set of all tuples t that satisfy $P(t)$

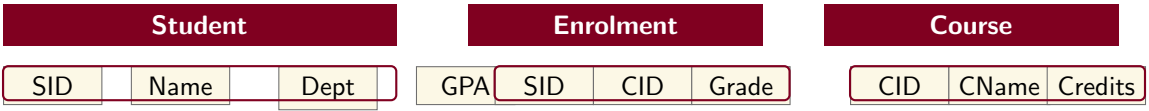
Atoms in $P(t)$:

- $t \in R$ (t is a tuple in relation R)
- $t.A\ \theta\ s.B$ (attribute comparison)
- $t.A\ \theta\ c$ (compare with constant c)
- $\theta \in \{=, \neq, <, \leq, >, \geq\}$

Connectives:

- $P_1 \wedge P_2$ (AND)
- $P_1 \vee P_2$ (OR)
- $\neg P$ (NOT)
- $\exists s \in R\,(P(s))$ (there exists)
- $\forall s \in R\,(P(s))$ (for all)

The Running Schema



Student(SID, Name, Dept, GPA)

Enrolment(SID, CID, Grade)

Course(CID, CName, Credits)

Sample Data Used in Examples

Student has rows with SID 1001–1005; Dept values: CS, EE, ME; GPA 2.9–4.0. Course C101 = “Intro to DB”, C102 = “Algorithms”.

Example 1 — Students with GPA > 3.5

Query: Retrieve all attributes of students whose GPA exceeds 3.5

$$\{\ t \mid t \in \text{Student} \ \wedge \ t.\text{GPA} > 3.5\ \}$$

Equivalent SQL:

```
SELECT *
FROM   Student
WHERE  GPA > 3.5;
```

Reading the TRC Expression

"Give me every tuple t such that t is in the Student relation **and** t's GPA attribute is greater than 3.5."

Example 2 — Names of Students in the CS Department

Query: Return only the Name of CS students

$$\{\ t.\text{Name} \mid t \in \text{Student} \wedge t.\text{Dept} = \text{`CS'}\ \}$$

Projection in TRC

Writing t.Name on the left of | restricts the result to just the Name attribute — this is how **projection** is expressed.

Equivalent SQL:

```
SELECT Name
FROM   Student
WHERE  Dept = 'CS';
```

Example 3 — Students Enrolled in Course C101

Query: Names of students who are enrolled in course C101

$$\{\ t.\text{Name} \mid t \in \text{Student} \ \wedge\ \exists e \in \text{Enrolment}\ (e.\text{SID} = t.\text{SID} \ \wedge\ e.\text{CID} = \text{`C101'})\ \}$$

Equivalent SQL:

```
SELECT s.Name
FROM   Student s
WHERE  EXISTS (
           SELECT 1
           FROM   Enrolment e
           WHERE  e.SID = s.SID
             AND  e.CID = 'C101'
       );
```

Existential Quantifier

$\exists e \in \text{Enrolment}$ means "there is at least one enrolment tuple e such that ...". This mirrors SQL's `EXISTS`.

Example 4 — Students Enrolled in *Every* Course

Query (Universal Quantifier): Students enrolled in all courses

$$\{\ t.\text{Name} \mid t \in \text{Student} \wedge \forall\, c \in \text{Course}\ (\exists\, e \in \text{Enrolment}\ (e.\text{SID} = t.\text{SID} \wedge e.\text{CID} = c.\text{CID}))\ \}$$

Universal Quantifier $\forall$

$\forall$ is trickier — for *every* course tuple c, there must exist an enrolment tuple linking this student to that course.

Equivalence with Division ($\div$)

This is the TRC equivalent of the relational algebra **division** operator: Student.SID $\div$ Course.CID via Enrolment.

Safe vs. Unsafe TRC Expressions

Problem: Infinite Results

TRC can express queries that produce *infinitely many* tuples — tuples that are not in any base relation.

Example (unsafe): $\{t \mid \neg(t \in \text{Student})\}$ — all tuples *not* in Student. This is infinite!

Safe Expression (Codd, 1972)

A TRC expression $\{t \mid P(t)\}$ is **safe** if every value that appears in the result is a value that already exists in the **domain** of P (i.e., it appears in some tuple of the referenced relations or as a constant in P).

Safe:

$$\{t \mid t \in \text{Student} \wedge t.\text{GPA} > 3.5\}$$

All returned tuples come from Student.

Unsafe:

$$\{t \mid \neg(t \in \text{Student})\}$$

Could include tuples never stored anywhere.

Domain of a Formula

Domain of P, written dom(P)

The set of all values that appear either:

1. As constants in P, **or**
2. In any tuple of any relation mentioned in P

Formal Safety Condition: Expression $\{t \mid P(t)\}$ is safe iff
(1) for every free variable x in P, all values x can take that satisfy P are in dom(P); and
(2) for every sub-formula $\exists\, u(Q(u))$ or $\forall\, u(Q(u))$, all values of u satisfying Q are in dom(Q).

Practical Rule

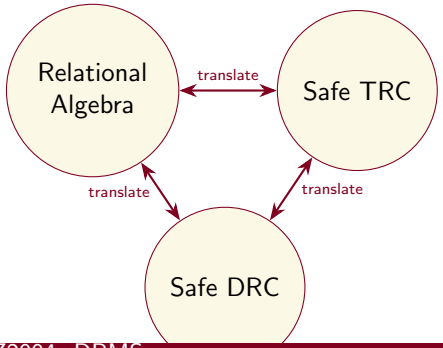
If every variable in your TRC query is *introduced* via a membership atom ($t \in R$) and stays within the relation's data, your expression is almost certainly safe.

Codd's Theorem — Equivalence of Query Languages

Codd's Theorem (1972)

$$\text{Relational Algebra} \equiv \text{Safe TRC} \equiv \text{Safe DRC}$$

The three formalisms are **equally expressive**: any query that can be written in one can be written in the other two.



Implications:

- SQL (based on RC) has the same power as RA
- "Relational completeness" benchmark: a query language is complete if it is at least as expressive as RA
- Unsafe expressions go *beyond* RA power (but produce infinite results)

RA $\leftrightarrow$ TRC Translation Examples

Relational Algebra	**Safe TRC equivalent**
$\sigma_{\text{GPA}>3.5}(\text{Student})$	$\{t \mid t \in \text{Student} \wedge t.\text{GPA} > 3.5\}$
$\pi_{\text{Name}}(\sigma_{\text{Dept}=\text{CS}}(\text{Student}))$	$\{t.\text{Name} \mid t \in \text{Student} \wedge t.\text{Dept} = \text{`CS'}\}$
$\text{Student} \bowtie_{\text{SID}} \text{Enrolment}$	$\{t \mid t \in \text{Student} \wedge \exists e \in \text{Enrolment}$ (e.SID=t.SID)$\}$

Translation Direction

RA $\rightarrow$ TRC: every RA operator maps to a logical connective or quantifier.
TRC $\rightarrow$ RA: safe TRC can always be mechanically converted back to RA.

Summary

Key Takeaways — L13 Tuple Relational Calculus

1. TRC expresses queries as $\{t \mid P(t)\}$ — declarative, not procedural
2. Tuple variable t ranges over entire tuples of a relation
3. Predicates use $\in$, attribute comparisons, $\wedge$, $\vee$, $\neg$, $\exists$, $\forall$
4. **Safe expression**: result contains only values from $\text{dom}(P)$ — avoids infinite output
5. **Codd's Theorem**: Safe TRC $\equiv$ RA $\equiv$ Safe DRC in expressive power
6. SQL is a practical implementation of relational calculus ideas

Next Lecture — L14: Domain Relational Calculus

L14 (31 Mar): Domain variables, DRC syntax, converting TRC to DRC, DRC vs TRC comparison, and an introduction to **QBE** (Query By Example).