Relational calculus and SQL

Agenda

• Relational Calculus
• Relational Calculus and SQL
  – More on Quantifiers
• Relational Calculus and Relational Algebra
  – Equivalence Rules
• Back to Integrity Constraints
  – Assertions
  – Triggers

Relational Calculus

• Although relational algebra is useful in the analysis of query evaluation, SQL is actually based on a different query language: relational calculus
• Relational calculus useful to define the semantics of the relational algebra and SQL
• There are two relational calculi:
  – Tuple relational calculus (TRC)
  – Domain relational calculus (DRC)

Tuple Relational Calculus

• Form of query:
  \{ T \mid \text{Condition}(T) \}
  – T is the target – a variable that ranges over tuples of values
  – Condition is the body of the query
  • Involves T (and possibly other variables)
  • Evaluates to true or false if a specific tuple is substituted for T

Query Result

The result of a TRC query with respect to a given database is the set of all choices of tuples for the variable T that make the query condition a true statement about the database

Tuple Relational Calculus: Example

\{ T \mid \text{Teaching}(T) \text{ AND } T.\text{Semester} = 'F2000' \}
• When a concrete tuple has been substituted for T:
  – Teaching(T) is true if T is in the relational instance of Teaching
  – T.\text{Semester} = 'F2000' is true if the semester attribute of T has value F2000
  – Equivalent to:
    
    SELECT *
    FROM Teaching T
    WHERE T.\text{Semester} = 'F2000'
Query Condition

• Atomic condition:
  – \( P(T) \), where \( P \) is a relation name
  – \( T.A \) oper \( S.B \) or \( T.A \) oper const, where \( T \) and \( S \) are relation names, \( A \) and \( B \) are attributes and \( \text{oper} \) is a comparison operator (e.g., =, , <, >, ∈, etc)

• (General) condition:
  – atomic condition
  – if \( C_1 \) and \( C_2 \) are conditions then \( C_1 \land C_2 \), \( C_1 \lor C_2 \), and \( \neg C_1 \) are conditions
  – if \( R \) is a relation name, \( T \) a tuple variable, and \( C(T) \) is a condition that uses \( T \), then \( \forall T \in R \,(C(T)) \) and \( \exists T \in R \,(C(T)) \) are conditions

Bound and Free Variables

• \( X \) is a free variable in the statement \( C_1 \): “\( X \) is in CS305”
  (this might be represented more formally as \( C_1(X) \))
  – The statement is neither true nor false in a particular state of the database until we assign a value to \( X \)

• \( X \) is a bound (or quantified) variable in the statement \( C_2 \): “there exists a student \( X \) such that \( X \) is in CS305”
  (this might be represented more formally as \( \exists X \in S \,(C_1(X)) \))
  where \( S \) is the set of all students
  – This statement can be assigned a truth value for any particular state of the database

Bound and Free Variables in TRC

• Bound variables are used to make assertions about tuples in database (used in conditions)
• Free variables designate the tuples to be returned by the query (used in targets)

\[
\{ S | \text{Student}(S) \land (\exists T \in \text{Transcript} \,(S.Id = T.StudId \land T.CrsCode = ‘CS305’)) \}
\]

  – When a value is substituted for \( S \) the condition has value true or false
  – There can be only one free variable in a condition (the one that appears in the target)

Example

\[
\{ E \mid \text{Course}(E) \land \\
\forall S \in \text{Student} ( \\
\exists T \in \text{Transcript} ( \\
\quad T.StudId = S.Id \land \\
\quad T.CrsCode = E.CrsCode ) ) \\
\}
\]

Returns the set of all course tuples corresponding to all courses that have been taken by all students

TRC Syntax Extension

We add syntactic sugar to TRC, which simplifies queries and make the syntax even closer to that of SQL

\[
\{ S.Name, T.CrsCode \mid \text{Student}(S) \land \text{Transcript}(T) \\
\quad \text{AND} \ldots \}
\]

instead of

\[
\{ R \mid \exists S \in \text{Student} \,(R.Name = S.Name) \\
\quad \text{AND} \exists T \in \text{Transcript} \,(R.CrsCode = T.CrsCode) \\
\quad \text{AND} \ldots \}
\]

where \( R \) is a new tuple variable with attributes \( \text{Name} \) and \( \text{CrsCode} \)

Agenda

• Relational Calculus
• Relational Calculus and SQL
  – More on Quantifiers
• Relational Calculus and Relational Algebra
  – Equivalence Rules
• Back to Integrity Constraints
  – Assertions
  – Triggers
Relation Between TRC and SQL

• Target T corresponds to SELECT list: the query result contains the entire tuple
• Body split between two clauses:
  – Teaching (T) corresponds to FROM clause
  – T.Semester = ‘F2000’ corresponds to WHERE clause

List the names of all professors who have taught MGT123
  – In TRC:
    \[
    \{ P \mid \text{Professor}(P) \land \exists T \in \text{Teaching} \ (P.Id = T.ProfId \land T.CrsCode = 'MGT123') \}
    \]
  – In SQL:
    \[
    \text{SELECT P.Name FROM Professor P, Teaching T WHERE P.Id = T.ProfId AND T.CrsCode = 'MGT123'}
    \]

Core of SQL is merely a syntactic sugar on top of TRC

What Happened to Quantifiers in SQL?

• SQL has no quantifiers: how come? It uses conventions:
  – Convention 1: Universal quantifiers are not allowed (but SQL:1999 introduced a limited form of explicit ∀)
  – Convention 2: Make existential quantifiers implicit: Any tuple variable that does not occur in SELECT is assumed to be implicitly quantified with ∃
• Compare:

  \[
  \begin{align*}
  &\{ P.Name \mid \text{Professor}(P) \land \exists T \in \text{Teaching} \ (… ) \} \\
  &\quad \quad \text{and} \\
  &\text{SELECT P.Name FROM Professor P, Teaching T} \\
  &\quad \quad \ldots \ldots \ldots \\
  \end{align*}
  \]

More on Quantification

• Adjacent existential quantifiers and adjacent universal quantifiers commute:
  – \( \exists T \in \text{Transcript} \ (\exists T1 \in \text{Teaching} \ (…)) \) is same as \( \exists T1 \in \text{Teaching} \ (\exists T \in \text{Transcript} \ (…)) \)
• Adjacent existential and universal quantifiers do not commute:
  – \( \exists T \in \text{Transcript} \ (\forall T1 \in \text{Teaching} \ (…)) \) is different from \( \forall T1 \in \text{Teaching} \ (\exists T \in \text{Transcript} \ (…)) \)

More on Quantification

• A quantifier defines the scope of the quantified variable (analogously to a begin/end block):
  \[
  \forall T \in R1 \ (U(T) \land \exists T \in R2(V(T)))
  \]
  is the same as:
  \[
  \forall T \in R1 \ (U(T) \land \exists S \in R2(V(S)))
  \]
• Universal domain: Assume a domain, \( U \), which is a union of all other domains in the database. Then, instead of \( \forall T \in U \) and \( \exists S \in U \) we simply write \( \forall T \) and \( \exists T \)
Views in TRC

- Problem: List students who took a course from every professor in the Computer Science Department
- Solution:
  - First create view: All CS professors
    $$\text{CSProf} = \{ \text{Professor} | \text{Professor(P) AND PDeptId = 'CS'} \}$$
  - Then use it
    $$\{ (\text{S.Id} | \text{Student(S)} \\text{AND} \\ \text{VP} \in \text{CSProf}) \text{AND} \text{Teaching} | \text{R} \in \text{Transcript} \text{AND} \text{P.Id} = \text{P.ProfId} \text{AND} \text{S.Id} = \text{R.StudId} \text{AND} \text{T.CrsCode} = \text{R.CrsCode} \text{AND} \text{T.Semester} = \text{R.Semester} \}$$

Queries with Implication

- Did not need views in the previous query, but doing it without a view has its pitfalls: need the implication → (if-then):
  $$\{ (\text{S.Id} | \text{Student(S)} \text{AND} \text{VP} \in \text{Professor(P)} \text{AND} \text{T.} \text{DeptId} = \text{'CS'} \text{AND} \text{T.} \text{Teaching} | \text{R} \in \text{Transcript} \text{AND} \text{P.Id} = \text{T.} \text{Professor} | \text{P.Id} \text{AND} \text{S.Id} = \text{R.Id} \text{AND} \text{T.} \text{CrsCode} = \text{R.} \text{CrsCode} \text{AND} \text{T.} \text{Semester} = \text{R.} \text{Semester} \}$$

- Why P.DeptId = 'CS' → … and not P.DeptId = 'CS' AND … ?

More complex SQL to TRC Conversion

Using views, translation between complex SQL queries and TRC is direct:

**SELECT R1.A, R2.C FROM R1, R2 WHERE condition(R1.R2) AND R1.B IN (SELECT R3.E FROM R1, R2 WHERE condition(R2,R3,R4))**

versus:


Relation Between Relational Algebra and TRC

- Consider the query {T | NOT Q(T)}: returns the set of all tuples not in relation Q
  - If the attribute domains change, the result set changes as well
  - This is referred to as a domain-dependent query
- Another example: {T | \(\forall S(R(S)) \text{V} Q(T)\)}
  - Try to figure out why this is domain-dependent
- Only domain-independent queries make sense, but checking domain-independence is undecidable
  - There are syntactic restrictions that guarantee domain-independence

Agenda

- Relational Calculus
- Relational Calculus and SQL
  - More on Quantifiers
- Relational Calculus and Relational Algebra
  - Equivalence Rules
- Back to Integrity Constraints
  - Assertions
  - Triggers

Relation Between Relational Algebra and TRC

- Relational algebra (but not TRC) queries are always domain-independent
  - Proven by induction
- TRC and relational algebra are equally expressive for domain-independent queries
  - Proving that every domain-independent TRC/DRC query can be written in the algebra is somewhat hard
  - We will show the other direction: that algebraic queries are expressible in TRC/DRC
Relationship between Relational Algebra and TRC

- Algebra: $\sigma_{\text{Condition}}(R)$
- TRC: $\{ T \mid R(T) \text{ AND Condition} \}$

Transformation Rules

- Selections:
  $\sigma_{\text{Cond}_1 \land \text{Cond}_2}(R) \equiv \sigma_{\text{Cond}_1}(\sigma_{\text{Cond}_2}(R))$
- Projections:
  $\pi_{\text{attr}}(R) \equiv \pi_{\text{attr}}(\pi_{\text{attr}'}(R))$, if \text{attr} \subseteq \text{attr}'
- Commute projection and selection:
  $\pi_{\text{attr}}(\sigma_{\text{Cond}}(R)) \equiv \sigma_{\text{Cond}}(\pi_{\text{attr}}(R))$, if \text{attr} \supseteq \text{all attributes in Cond}

Pushing Selections and Projections

- $\sigma_{\text{Cond}}(R \times S) \equiv R \bowtie_{\text{Cond}} S$
  - \text{Cond} relates attributes of both R and S
- $\sigma_{\text{Cond}}(R \times S) \equiv \sigma_{\text{Cond}}(R) \times S$
  - \text{Cond} involves only the attributes of R
- $\pi_{\text{attr}}(R \times S) \equiv \pi_{\text{attr}}(\pi_{\text{attr}'}(R) \times S)$, if attributes(R) \supseteq \text{attr}' \supseteq \text{attr} \cap \text{attributes}(R)$

Commutativity and Associativity of Join
(and Cartesian Product as Special Case)

- Join commutativity:
  $R \bowtie S \equiv S \bowtie R$
- Join associativity:
  $R \bowtie (S \bowtie T) \equiv (R \bowtie S) \bowtie T$

Equivalence Example

- $\sigma_{\text{Cond}_1 \land \text{Cond}_2 \land \text{Cond}_3}(R \times S) \equiv \sigma_{\text{Cond}_1}(\sigma_{\text{Cond}_2}(\sigma_{\text{Cond}_3}(R \times S)))$
  $\equiv \sigma_{\text{Cond}_1}(\sigma_{\text{Cond}_2}(R \times \sigma_{\text{Cond}_3}(S)))$
  $\equiv \sigma_{\text{Cond}_2}(\sigma_{\text{Cond}_1}(R) \bowtie_{\text{Cond}_3} \sigma_{\text{Cond}_3}(S))$

assuming \text{Cond}_2 involves only attributes of R, \text{Cond}_3 involves only attributes of S, and \text{Cond}_1 relates attributes of R and S
Agenda

- Relational Calculus
- Relational Calculus and SQL
- Relational Calculus and Relational Algebra
- Back to Integrity Constraints
  - Assertions
  - Triggers

Assertion

- Element of schema (like table)
- Symmetrically specifies an inter-relational constraint
- Applies to entire database (not just the individual rows of a single table)
  - hence it works even if Employee is empty
  - CREATE ASSERTION Don’tFireEveryone
  - CHECK (0 < SELECT COUNT(*) FROM Employee)

Assertion

CREATE ASSERTION KeepEmployeeSalariesDown
CHECK (NOT EXISTS(
  SELECT * FROM Employee E
  WHERE E.Salary > E.MgrSalary))

Assertion

CREATE ASSERTION NoEmptyCourses
CHECK (NOT EXISTS (
  SELECT * FROM Teaching T
  WHERE -- for each row T check
    -- the following condition
    NOT EXISTS (
      SELECT * FROM Transcript R
      WHERE T.CrsCode = R.CrsCode
      AND T.Semester = R.Semester)
))

Assertion and Inclusion Dependency

Triggers

- A general mechanism for handling events, i.e., transitions from one database state to another
  - Not in SQL-92, but is in SQL:1999
- Trigger is a schema element (like table, assertion, …)
  - CREATE TRIGGER CrsChange
  - AFTER UPDATE OF CrsCode, Semester ON Transcript
  - WHEN (Grade IS NOT NULL)
  - ROLLBACK

Trigger Overview

- Element of the database schema
- General form:
  - ON <event> IF <condition> THEN <action>
  - <Event> - request to execute database operation
  - <Condition> - predicate evaluated on database state
  - <Action> - execution of procedure that might involve database updates
- Example:
  - ON updating maximum course enrollment
  - IF number registered > new max enrollment limit
  - THEN deregister students using LIFO policy
Trigger Details

- **Activation** - Occurrence of the event
- **Consideration** - The point, after activation, when condition is evaluated
  - Immediate or deferred (when the transaction requests to commit)
  - Condition might refer to both the state before and the state after event occurs

Trigger Details

- **Execution** – point at which action occurs
  - With deferred consideration, execution is also deferred
  - With immediate consideration, execution can occur immediately after consideration or it can be deferred
    - If execution is immediate, execution can occur before, after, or instead of triggering event.
    - Before triggers adapt naturally to maintaining integrity constraints; violation results in rejection of event.

Trigger Details

- **Granularity**
  - Row-level granularity: change of a single row is an event (a single UPDATE statement might result in multiple events)
  - Statement-level granularity: events are statements (a single UPDATE statement that changes multiple rows is a single event).

Trigger Details

- **Multiple Triggers**
  - How should multiple triggers activated by a single event be handled?
    - Evaluate one condition at a time and if true immediately execute action or
    - Evaluate all conditions, then execute actions
  - The execution of an action can affect the truth of a subsequently evaluated condition so the choice is significant.

Summary

- Relational calculus (declarative - what) useful to determine the semantics of the relational algebra (procedural - how)
  - The relational calculus and the relational algebra are actually equivalent
  - Key mapping between declarative and procedural languages
- The core of the SQL query language is syntactic sugar on top of the Tuple Relational Calculus
- SQL query language also used for:
  - Defining a valid state of the database (assertions)
  - Defining valid transitions between states (triggers)