Week 3: Relational Data Model & Relational Algebra

Lecturer: Dr. Thomas Roelleke

Room CS423

Lecture Outline

- History
- Description
- Definitions
- Characteristics
- Constraint
- Relational Algebra (RA)
- Examples

History

- Introduced by Codd in 1970 and provides:
  - a simple data structure for modelling all data
  - mathematically based
  - a standard for implementing data models

- Consequences/Implications:
  - Simplicity means that correctness is easier to establish
  - Standardisation means that distributed data can be combined more easily
  - Improvements to the facilities and the implementation can be shared easily

Description of the Relational Model

- All the information stored in a Relational Database is held in relations.
- No other data structures!
- A relation may be thought of as a table.

STUDENT

name	matric	exam1	exam2
Mounia	891023	12	58
Jane	892361	66	90
Thomas	880123	50	65

- A relation has:
  - a **name**
  - an unchanging set of **columns (attributes);** named and typed
  - a time varying set of **rows (tuples)**

Definitions (1)

- An **attribute** is a column of a relation:
  - a **name**: the role the column has in this relation
  - a **domain**: the set of values it may take

- A **domain** is a set of atomic values (indivisible):
  - its **meaning** e.g. the set of matriculation numbers
  - its **format** e.g. a 6 digit integer - a range

    e.g. the days of the week - a set of values

- A tuple is a row of a relation:
  - a set of values which are instances of the attributes of a relation

Definitions (2)

- Relational schema:
  - a set of attributes and is written R (A_1, $A_{2,...}$ A_n)
  - e.g. STUDENT (name, matric, exam1, exam2)

- Relation (instance of a relation):
  - a set of tuples which accords with some relational schema

- The degree of a relation (also referred to as the arity of a relation):
  - the number of attributes
  - (*Note*: "degree" in the ERM was what?)

- The cardinality:
  - the number of tuples
  - (*Note*: "cardinality" in the ERM was what?)

Definitions (3)

- Keys, or candidate Keys:
  - any set of attributes which are unique for each row

- Primary key:
  - one candidate key is chosen to identify the tuples
  - it is <u>underlined (what exactly is underlined?)</u>
  - e.g. STUDENT (name, <u>matric</u>, exam1, exam2)

Definitions (4)

- Relational database schema:
  - set of relation schemas together with a set of "integrity constraints"
- Relational database instance:
  - a set of relations realising a relational database schema
- Base relation:
  - relation which actually exists as a stored file
  - (vs. **temporary** or **view** relations)
- Foreign Key:
  - an attribute or set of attributes which match the primary key of another relation and thus link them

Characteristics of the Relational Model (1)

- No duplicate tuples - as the tuples are a **set**
  - Must be checked when:
    - a new tuple is added
    - a value is modified
    - a new relation is created as a projection of an old one
  - Implies that a primary key **always** exists (at worst all of the attributes)

- The tuples are unordered - again a property of sets
  - A table is only a representation of a relation
  - But, a physical storage of a relation will have an order

Characteristics of the Relational Model (2)

- The attributes are also unordered
  - **set** of attributes
  - no notion of getting the first attribute, next attribute, etc. (also no first tuple, next tuple, etc.)

- All values are atomic

S	PQ
S1	{(P1,200),(P2,300)}

must become →

S	P	Q
S1	P1	200
S1	P2	300

(First Normal Form - Normalization)

Characteristics of the Relational Model (3)

- Unknown values must be represented
  - These are replaced by **null** - a distinguished value
  - Research into different kinds of nulls "Don't Know", "Don't Care”, "Not Applicable" has proved difficult
  - Three-valued logic: true, false, unknown (open world assumption)
  - Reasoning over unknown is difficult

Query: List all students who are **NOT** considering a job in the academic sector. How will you model the relationship and how will you express the query?

Constraints (1)

- Constraints:
  - A set of rules which must hold for all relations in a DB and are enforced by the DBMS
- Key Constraints
  - Every candidate key (value of the key) is unique for every tuple
- Entity Integrity Constraint
  - No primary key (value of the key) of a base relation may be null
- Why?
  - The primary key acts as an identifier for the objects in the relation
  - A null key implies the existence of unidentifiable objects

Constraints (2)

- Referential Integrity
  - Any attributes of a relation which are Foreign Keys to another relation must take values which either
    - exist in the second relation
    - or are null

TEACHES

Name	Course
ML	229
RW	230
RS	231
JR	NULL

COURSE

Number	Other Attributes
229	
230	
232	

(Dangling Pointer: where does 231 point to?)

Enforcing Constraints

- The DBMS must continually check that constraints are not violated every time an update (insertion, deletion, modification) occurs
- Two strategies:
  - **refuse** to perform violating update
  - **compensate** in some way
- Compensation is performed by:
  - **Cascading** - make a compensating change to the current tuple then check everything that refers to it
  - **Restricting** - only change those tuples which do not violate constraints
  - **Nullifying** – set Foreign Keys to null if referential integrity is violated

Example: Deletion of Tuples

- Deletions can violate referential integrity by removing primary keys

COURSE

Number	Other Attributes
229	

TEACHES

Name	Course
ML	229

- Removing all courses starting "22" will leave a dangling foreign key
  - Restrict: reject and inform the user why
  - Cascade: remove the courses and any tuple in TEACHES with dangling foreign key (and pointers to entries in TEACHES and so on)
  - Restrict: remove only those courses which are not referred to
  - Nullify: update dangling pointers in the reference to NULL

Manipulating a Relational Database

- Changes - updates
- Retrieval - queries

(Sometimes the term query is used to mean either an update or a query)

- Three standard ways of doing this:
  - Two formal "languages":
    - **Relational Algebra** - allows the description of queries as a series of operations
    - **Relational Calculus** - allows the description of the desired result
    - RA: procedural; calculus: descriptive
  - One artificial (“real”) language:
    - **SQL** - the standard relational database manipulation language

Relational Algebra (1)

- Extract from the database a subset of the information which answers some question:
    - "What are the department names?"
    - "Tell me all the data held about male employees."
    - "What are the names of the employees in the R&D department?"

- Extraction consists of programs built out of:
    - retrieving part of some relation
    - linking two relations together in a meaningful way

Relational Algebra (2)

- Set of operations which can be combined to provide the result of a query in the form of a relation
- The algebra:
  - A collection of operations of two categories:
    - Special Relational Operations
    - Traditional Set Operations
  - A "relational assignment" statement so that partial results can be assigned a name
  - Renaming: change attribute names
- Querying process:
  - A sequence of operation calls of the form:

    newRelation := op(parameters including relation names, column names and conditions)
  - Usually does **not** create a copy of the data

Relational Algebra Operations

- Principal relation operations:
  - **select** - pick rows from a relation by some condition
  - **project** - pick columns by name
  - **join** - connect two relations usually by a Foreign Key
- Set operations:
  - **union** - make the table containing all the rows of two relations
  - **intersection** - pick the rows which are common to two relations
  - **difference** - pick the rows which are in one table but not another
  - **Cartesian product** - pair off each of the tuples in one relation with those in another - creating a double sized row for each pair

All of the operations return relations

Example Schema for RA Expressions

DEPARTMENT (<u>Dnum</u>, Dname, Manager)

EMPLOYEE (Name, City, <u>NI</u>, Dept, Salary, Sex, Age)

PROJECT (Pname, Plocation, Dnum, <u>Pnumber</u>)

WorksOn (<u>ENI</u>, <u>P</u>)

Selection

- Extract the tuples of a relation which satisfy some condition on the values of their rows and return these as a relation

LOCALS := σ (EMPLOYEE, CITY = ”LONDON")

- Syntax (there are different options):

  σ (RelationName, Condition)

where the condition can contain:

literals
column names
comparison operators (=, >, etc.)
boolean operators (and, not, or)

LONDONorYOUNGRICH := σ (EMPLOYEE,
CITY = “LONDON” or (SALARY > 60K and AGE < 30))

Projection

- Extracts some of the columns from a relation

  SexSalary := Π (EMPLOYEE, (SEX, SALARY))

  - No attribute may occur more than once
  - Duplicate will be removed

- Projection and selection combined

  Π (σ (EMPLOYEE, CITY = “LONDON”), (NAME,NI))

  - This does a selection followed by a projection
  - The DBMS may re-organise this for faster retrieval

Union

- Produce a relation which combines two relations by containing all of the tuples from each - removing duplicates
- The two relations must be "union compatible", i.e. have the same number of attributes drawn from the same domain (but maybe having different names)

LondonOrRich: = σ (EMPLOYEE, CITY = "LONDON") ∪
σ (EMPLOYEE, SALARY > 60K)

If attribute names differ, the names from the first one are taken

Same can be done using disjuncts!

Intersection

- Similar to union but returns tuples that are in both relations

$$\textbf{FemalesInLondon} := \sigma\,(\text{EMPLOYEE, CITY} = \text{“LONDON”}) \cap \sigma\,(\text{EMPLOYEE, SEX} = \text{“F”})$$

Try to express the query with a conjunct in the condition!

Difference

- Similar to union (and intersection) regarding schema constraints
- Returns tuples that are in the first relation but not the second

NonLocals := EMPLOYEE - LOCALS

- Intersection and difference both require union compatibility
- Intersection and difference use column names from the first relation

Cartesian Product

- The Cartesian Product of two relations A (a tuples) and B (b tuples) , which have attributes A_1 ... A_m and B_1 ... B_n is the relation with $m + n$ attributes containing row for every pair of rows one from A and one from B. The result has $a \times b$ tuples

EMPLOYEE

name	NI	dept.
ML	123	5
JR	456	5

DEPENDENT

ENI	name	sex
123	SM	M
123	TC	F
456	JA	F

EMPLOYEE × DEPENDENT

	name	NI	dept.	ENI	name	sex
*	ML	123	5	123	SM	M
*	ML	123	5	123	TC	F
	ML	123	5	456	JA	F
	JR	456	5	123	SM	M
	JR	456	5	123	TC	F
*	JR	456	5	456	JA	F

Cartesian Product

- Fairly meaningless as it stands and is rarely used on its own
    - More meaningful is the subset of rows marked with a star

- This could be created with the following selection:

σ (EMPLOYEE x DEPENDENT, NI = ENI)

The selection essentially makes use of the foreign key

- Cartesian product followed by this kind of selection is called a join because it joins together two relations

Join

- The join of relations A and B pairs off the tuples of A and B so that named attributes from the relations have the same value

The common column then appears just once

EMPLOYEE

Name	NI	Dept
RLC	123	5
MPA	456	5
RCW	345	7

DEPARTMENT

Dnum	Dname	Manager
5	R&D	456
6	Production	111
7	Admin	345

Join is written:

⊗ (EMPLOYEE, Dept, DEPARTMENT, Dnum)

These two relations can be joined in two ways

Join

- Joining the relations on EMPLOYEE.Dept = DEPARTMENT.Dnum puts together an employee's record with that of the department he or she works in:

EMPLOYEE-DEPARTMENT

Name	NI	Dept	Dname	Manager
RLC	123	5	R&D	456
MPA	456	5	R&D	456
RCW	345	7	Admin	345

- Joining the relations on EMPLOYEE.NI = DEPARTMENT.Manager puts a department 's record together with that of the employee who manages it:

EMPLOYEE-DEPARTMENT

Name	NI	Dept	Dnum	Dname
MPA	456	5	5	R&D
RCW	345	7	7	Admin

Unmatched tuples disappear - example "RLC" and "Production"

There are other forms which perform differently - this one is called **natural join**

Division

- "Give the NI of employees who work on ***all*** projects that John Smith works on"
  - find all of the numbers of projects that John Smith works on; assume these are projects 3 and 4
  - examine the WorksOn table (with the hours attribute removed) and return all the employee numbers which are paired

JSPNos

PNo
3
4

WorksOn

NI	PNo
123	1
145	3
145	4
169	1
169	3
172	2
172	3
172	4

Result

NI
145
172

Result := WorksOn ÷ JSPnos

R ($r_1,\ldots r_m, c_1, \ldots c_n$) ÷ S($c_1, \ldots c_n$) returns the relation with attributes ($r_1,\ldots r_m$) such that each tuple in the result appears once in R associated with every tuple in S

Example Schema for RA Expressions

DEPARTMENT (Dnumber, Dname, Manager)

EMPLOYEE (Name, Address, NI, Dept, DateOfBirth)

PROJECT (Pname, Plocation, Dnum, Pnumber)

WorksOn (ENI, P)

Examples of RA Expressions

Give the names and addresses of all employees who work for the R&D department

ResDept := σ (DEPARTMENT, Dname = "R&D")

this should return just one tuple if …?

ResDeptEmps := ⊗ (ResDept, Dnumber, EMPLOYEE, Dept)

this picks out the employees in that department; join[Dnumber=Dept](ResDept, EMPLOYEE)

Result := Π (ResDeptEmps, (Name, Address))

Notes:

- The order could be changed to make it faster
- The processing steps could be expressed in nested form as:

  Π (⊗ (σ (DEPARTMENT, Dname = "R&D"),
  Dnumber, EMPLOYEE, Dept), (Name, Address))

Examples of RA Expressions

List the name, controlling department name, the department manager's name, address and date of birth for every project in Stafford

StaffordProjs := σ (PROJECT, PLocation = "Stafford")

it is common to start by restricting to an area of interest

StaffordProjDepts:= ⊗ (StaffordProjs, Dnum, DEPARTMENT, Dnumber)

this brings together the department information

StaffordProjDeptManagers:= ⊗ (StaffordProjDepts, Manager, EMPLOYEE, NI)

this brings in the manager information

Result := Π (StaffordProjDeptManagers, Pname, Dname, Name, Address, DateOfBirth)

Examples of RA Expressions

List the names of employees who work on all the projects controlled by department 5

Dept5ProjNumbers := Π (σ (PROJECT, Dnum = 5), Pnumber)

EmpProj := Π (WorksOn, ENI, P)

remove the hours column

EmpNIs := EmpProj ÷ Dept5ProjNumbers

Employees := ⊗ (EmpNIs, ENI, EMPLOYEE, NI)

bring in the rest of the employee information

Result := Π (Employees, Name)

Note on Syntax

- There is no standard syntax for the relational algebra
- As far as these slides are concerned the following expressions were used:

  ⊗ (WorksOn, ENI, EMPLOYEE, NI)

  Equiv to

  join (WorksOn, ENI, EMPLOYEE, NI)

  Equiv to

  join (WorksOn, EMPLOYEE, ENI=NI)

  Equiv to

  Join[ENI=NI](WorksOn, EMPLOYEE)

  If it is clear, then it is good.

Summary

- RA is the core of a DB system
- Main operators:
  - Unary: Select and Project
  - Basic and Binary: Union, Difference (Subtraction), Product
  - Composed and Binary: Intersection, Join, Division
- RA optimisation
- Advanced topic: Probabilistic Relational Algebra
  - See Fuhr/Roelleke, ACM TOIS 1997