Relational Data Model

Introduction

- Proposed by Edgar F Codd (1923-2003) in the early seventies [Turing Award – 1981]
- Most of the modern DBMS use the *relational* data model.
- Simple and elegant model with a mathematical basis.
- Led to the development of a theory of data dependencies and database design.
- Relational algebra operations – crucial role in query optimization and execution.
- Laid the foundation for the development of
  - Tuple relational calculus and then
  - Database standard SQL

Relation Scheme

- Consists of relation name, and a set of attributes or field names or column names. Each attribute has an associated domain.
- *Example:*

```
student ( studentName     : string,
          rollNumber      : string,
          phoneNumber     : integer,
          yearOfAdmission : integer,
          branchOfStudy   : string )
```

Relation name

Attribute names

domains

- *Domain* – set of *atomic* (or *indivisible*) values – data type

Relation Instance

- A finite *set of tuples* constitute a relation instance.
- A tuple of the relation with scheme $R = (A_1, A_2, \ldots, A_m)$ is an ordered sequence of values $(v_1, v_2, \ldots, v_m)$ such that $v_i \in$ domain (A_i), $1 \leq i \leq m$

student

studentName	rollNumber	yearOf Admission	phoneNumber	branch Of Study
Ravi Teja	CS05B015	2005	9840110489	CS
Rajesh	CS04B125	2004	9840110490	CS
		⋮		

No duplicate tuples (or rows) in a relation instance.

We shall later see that in SQL, duplicate rows would be allowed in tables.

Another Relation Example

enrollment (studentName, rollNo, courseNo, sectionNo)

enrollment

studentName	rollNumber	courseNo	sectionNo
Rajesh	CS04B125	CS3200	2
Rajesh	CS04B125	CS3700	1
Suresh	CS04B130	CS3200	2
		⋮	

Keys for a Relation (1/2)

- **Key**: A set of attributes K, whose values uniquely identify a tuple in <u>any</u> instance. And none of the proper subsets of K has this property

  Example: {*rollNumber*} is a key for *student* relation.

  {*rollNumber, name*} – values can uniquely identify a tuple
  - but the set is not *minimal*
  - not a Key

- A key can not be determined from any particular instance data
  - it is an intrinsic property of a scheme
  - it can only be determined from the meaning of attributes

Keys for a Relation (2/2)

- A relation can have more than one key.
- Each of the keys is called a *candidate* key

  Example: *book* (*isbnNo, authorName, title, publisher, year*)

  (Assumption : books have only one author)

  Keys: {*isbnNo*}, {*authorName, title*}
- A relation has at least one key

  - the set of all attributes, in case no proper subset is a key.
- **Superkey**: A set of attributes that contains a key as a subset.
  - A key can also be defined as a *minimal superkey*
- **Primary Key**: One of the candidate keys chosen for indexing purposes (More details later…)

Relational Database Scheme and Instance

Relational database scheme: *D* consist of a finite no. of relation schemes and a set *I* of integrity constraints.

Integrity constraints: Necessary conditions to be satisfied by the data values in the relational instances so that the set of data values constitute a meaningful database

- domain constraints
- key constraints
- referential integrity constraints

Database instance: Collection of relational instances satisfying the integrity constraints.

Domain and Key Constraints

- **Domain Constraints**: Attributes have associated domains
  Domain – set of atomic data values of a specific type.
  Constraint – stipulates that the actual values of an attribute in any tuple must belong to the declared domain.

- **Key Constraint**: Relation scheme – associated keys
  Constraint – if *K* is supposed to be a key for scheme *R*, *any* relation instance *r* on *R* should not have two tuples that have identical values for attributes in *K*.
  Also, none of the key attributes can have null value.

Foreign Keys

- Tuples in one relation, say $r_1(R_1)$, often need to refer to tuples in another relation, say $r_2(R_2)$
  - to capture relationships between entities
- Primary Key of R_2 : $K = \{B_1, B_2, \ldots, B_j\}$
- A set of attributes $F = \{A_1, A_2, \ldots, A_j\}$ of R_1 such that
  $\mathrm{dom}(A_i) = \mathrm{dom}(B_i), \quad 1 \le i \le j$ and
  whose values are used to refer to tuples in r_2
  is called a *foreign key* in R_1 *referring* to R_2.

- R_1, R_2 can be the same scheme also.
- There can be more than one foreign key in a relation scheme

Foreign Key – Examples (1/2)

Foreign key attribute *deptNo* of *course* relation refers to
Primary key attribute *deptID* of *department* relation

Course

courseId	name	credits	deptNo
CS635	ALGORITHMS	3	1
CS636	A.I	4	1
ES456	D.S.P	3	2
ME650	AERO DYNAMICS	3	3

Department

deptId	name	hod	phone
1	COMPUTER SCIENCE	CS01	22576235
2	ELECTRICAL ENGG	ES01	22576234
3	MECHANICAL ENGG	ME01	22576233

Foreign Key – Examples(2/2)

It is possible for a foreign key in a relation
to refer to the primary key of the relation itself

An Example:

univEmployee (<u>empNo</u>, name, sex, salary, dept, reportsTo)

reportsTo is a foreign key referring to *empNo* of the same relation

Every employee in the university reports to some other employee for administrative purposes
- except the *vice-chancellor*, of course!

Referential Integrity Constraint (RIC)

- Let F be a foreign key in scheme R_1 referring to scheme R_2 and let K be the primary key of R_2.
- **RIC**: any relational instances r_1 on R_1 and r_2 on R_2 must be s.t for any tuple t in r_1, either its F-attribute values are all *null* or they are identical to the K-attribute values of *some* tuple in r_2.
- RIC ensures that references to tuples in r_2 are for *currently existing* tuples.
  - That is, there are no *dangling* references.

Referential Integrity Constraint (RIC) - Example

COURSE

courseId	name	credits	deptNo
CS635	ALGORITHMS	3	1
CS636	A.I	4	1
ES456	D.S.P	3	2
ME650	AERO DYNAMICS	3	3
CE751	MASS TRANSFER	3	4

DEPARTMENT

deptId	name	hod	phone
1	COMPUTER SCIENCE	CS01	22576235
2	ELECTRICAL ENGG.	ES01	22576234
3	MECHANICAL ENGG.	ME01	22576233

The new course refers to a non-existent department and thus violates the RIC

Example Relational Scheme

student (<u>rollNo</u>, name, degree, year, sex, deptNo, advisor)
degree is the program (B Tech, M Tech, M S, Ph D etc) for which the student has joined.
year is the year of admission and
advisor is the EmpId of a faculty member identified as the student's advisor.

department (<u>deptId</u>, name, hod, phone)
phone is that of the department's office.

professor (<u>empId</u>, name, sex, startYear, deptNo, phone)
startYear is the year when the faculty member has joined the department *deptNo*.

Example Relational Scheme

course (<u>courseId</u>, cname, credits, deptNo)
deptNo indicates the department that offers the course.

enrollment (<u>rollNo, courseId, sem, year</u>, grade)
sem can be either "odd" or "even" indicating the two semesters of an academic year.
The value of *grade* will be null for the current semester and non-null for past semesters.

teaching (<u>empId, courseId, sem, year</u>, classRoom)

preRequisite (<u>preReqCourse, courseID</u>)
Here, if (c1, c2) is a tuple, it indicates that c1 should be successfully completed before enrolling for c2.

Example Relational Scheme

student (rollNo, name, degree, year, sex, deptNo, advisor)

department (deptId, name, hod, phone)

professor (empId, name, sex, startYear, deptNo, phone)

course (courseId, cname, credits, deptNo)

enrollment (rollNo, courseId, sem, year, grade)

teaching (empId, courseId, sem, year, classRoom)

preRequisite (preReqCourse, courseId)

queries-1
queries-2
queries-3
XProd
TCQuery

Example Relational Scheme with RICs shown

student (rollNo, name, degree, year, sex, deptNo, advisor)

department (deptId, name, hod, phone)

professor (empId, name, sex, startYear, deptNo, phone)

course (courseId, cname, credits, deptNo)

enrollment (rollNo, courseId, sem, year, grade)

teaching (empId, courseId, sem, year, classRoom)

preRequisite (preReqCourse, courseId)

Relational Algebra

- A set of operators (unary and binary) that take relation instances as arguments and return new relations.
- Gives a procedural method of specifying a retrieval query.
- Forms the core component of a relational query engine.
- SQL queries are internally translated into RA expressions.
- Provides a framework for query optimization.

RA operations: *select* (σ), *project* (π), *cross product* ($\times$), *union* ($\cup$), *intersection* ($\cap$), *difference* ($-$), *join* ($\bowtie$)

The *select* Operator

- Unary operator.
- can be used to *select* those tuples of a relation that satisfy a given condition.
- *Notation*: $\sigma_\theta (r)$

  σ : select operator (read as *sigma*)

  θ : selection condition

  r : relation name
- Result: a relation with the same schema as r consisting of the tuples in r that satisfy condition θ
- Select operation is commutative:

  $\sigma_{c1} (\sigma_{c2} (r)) = \sigma_{c2} (\sigma_{c1} (r))$

Selection Condition

- *Select condition*:
  Basic condition or Composite condition
- *Basic condition*:
  Either A_i <compOp> A_j or A_i <compOp> c
- *Composite condition*:
  Basic conditions combined with logical operators AND, OR and NOT appropriately.
- Notation:

  <compOp> : one of $<, \leq, >, \geq, =, \neq$

  A_i, A_j : attributes in the scheme R of r

  c : constant of appropriate data type

Examples of *select* expressions

1. Obtain information about a professor with name "Giridhar"

   $\sigma_{\text{name = "Giridhar"}}$ (professor)

2. Obtain information about professors who joined the university between 1980 and 1985, both inclusive

   $\sigma_{\text{startYear} \geq 1980 \wedge \text{startYear} \leq 1985}$ (professor)

The *project* Operator

- Unary operator.
- Can be used to keep only the required attributes of a relation instance and throw away others.
- *Notation*: $\pi_{A_1, A_2, \ldots, A_k}(r)$ where $A_1, A_2, \ldots, A_k$ is a list L of desired attributes in the scheme of r.
- Result = $\{ (v_1, v_2, \ldots, v_k) \mid v_i \in dom(A_i),\ 1 \le i \le k$ and there is some tuple t in r *s.t* $t.A_1 = v_1, t.A_2 = v_2, \ldots, t.A_k = v_k \}$
- If $r_1 = \pi_L(r_2)$ then scheme of r_1 is L

Examples of *project* expressions

student

rollNo	name	degree	year	sex	deptNo	advisor
CS04S001	Mahesh	M.S	2004	M	1	CS01
CS03S001	Rajesh	M.S	2003	M	1	CS02
CS04M002	Piyush	M.E	2004	M	1	CS01
ES04M001	Deepak	M.E	2004	M	2	ES01
ME04M001	Lalitha	M.E	2004	F	3	ME01
ME03M002	Mahesh	M.S	2003	M	3	ME01

$\pi_{\text{rollNo, name}}(\text{student})$

rollNo	name
CS04S001	Mahesh
CS03S001	Rajesh
CS04M002	Piyush
ES04M001	Deepak
ME04M001	Lalitha
ME03M002	Mahesh

$\pi_{\text{name}}(\sigma_{\text{degree} = \text{“M.S”}}(\text{student}))$

name
Mahesh
Rajesh

Note: Mahesh is displayed only once because project operation results in a set.

Size of *project* expression result

- If $r_1 = \pi_L(r_2)$ then scheme of r_1 is L
- What about the number of tuples in r_1?
- Two cases arise:
  - Projection List L contains some key of r_2
    - Then $|r_1| = |r_2|$
  - Projection List L does not contain any key of r_2
    - Then $|r_1| \le |r_2|$

Set Operators on Relations

- As relations are sets of tuples, set operations are applicable to them; but not in all cases.
- **Union Compatibility** : Consider two schemes R_1, R_2 where $R_1 = (A_1, A_2, \ldots, A_k)$; $R_2 = (B_1, B_2, \ldots, B_m)$
- R_1 and R_2 are called *union-compatible* if
  - $k = m$ and
  - $dom(A_i) = dom(B_i)$ for $1 \le i \le k$
- **Set operations** – *union, intersection, difference*
  - Applicable to two relations if their schemes are union-compatible
- If $r_3 = r_1 \cup r_2$, scheme of r_3 is R_1 (as a convention)

Set Operations

r_1 - relation with scheme R_1
r_2 - relation with scheme R_2 - union compatible with R_1

$$r_1 \cup r_2 = \{t \mid t \in r_1 \text{ or } t \in r_2\}$$
$$r_1 \cap r_2 = \{t \mid t \in r_1 \text{ and } t \in r_2\}$$
$$r_1 - r_2 = \{t \mid t \in r_1 \text{ and } t \notin r_2\}$$

By convention, in all the cases, the scheme of the result is that of the first operand i.e r_1.

Cross product Operation

r_1	A_1	A_2	...	A_m
	a_{11}	a_{12}	...	a_{1m}
	a_{21}	a_{22}	...	a_{2m}
	.			
	a_{s1}	a_{s2}	...	a_{sm}

r_2	B_1	B_2	...	B_n
	b_{11}	b_{12}	...	b_{1n}
	b_{21}	b_{22}	...	b_{2n}
	.			
	b_{t1}	b_{t2}	...	b_{tn}

$r_1 \times r_2$

A_1	A_2	...	A_m	B_1	B_2	...	B_n
a_{11}	a_{12}	...	a_{1m}	b_{11}	b_{12}	...	b_{1n}
a_{11}	a_{12}	...	a_{1m}	b_{21}	b_{22}	...	b_{2n}
:				:			
a_{11}	a_{12}	...	a_{1m}	b_{t1}	b_{t2}	...	b_{tn}
a_{21}	a_{22}	...	a_{2m}	b_{11}	b_{12}	...	b_{1n}
a_{21}	a_{22}	...	a_{2m}	b_{21}	b_{22}	...	b_{2n}
:				:			
a_{21}	a_{22}	...	a_{2m}	b_{t1}	b_{t2}	...	b_{tn}
.							

r_1 : s tuples r_2 : t tuples $r_1 \times r_2$: $s \times t$ tuples

Example Query using *cross product*

Obtain the list of professors (Id and Name) along with the *name* of their respective departments
- Info is present in two relations − professor, department

Schema

- profDetail (eId, pname, deptno) ← $\pi_{\text{empId, name, deptNo}}$ (professor)
- deptDetail (dId, dname) ← $\pi_{\text{deptId, name}}$ (department)
- profDept ← profDetail × deptDetail
- desiredProfDept ← $\sigma_{\text{deptno = dId}}$ (profDept)
- result ← $\pi_{\text{eId, pname, dname}}$ (desiredProfDept)

Query using *cross product* − use of renaming

Query: Obtain the list of professors (Id and Name) along with the *name* of their respective departments

- profDetail (eId, pname, deptno) ← $\pi_{\text{empId, name, deptNo}}$ (professor)
  - this is a temporary relation to hold the intermediate result
  - "empId, name, deptNo" are being renamed as "eId, pname, deptno"
  - creating such relations helps us understand/formulate the query
  - we use "←" to indicate assignment operation.
- deptDetail (dId, dname) ← $\pi_{\text{deptId, name}}$ (department)
  - another temporary relation
- Renaming is necessary to ensure that the cross product has distinct attribute names.

Use of renaming operator ρ

Query: Obtain the list of professors (Id and Name) along with the *name* of their respective departments

- One can use the rename operator ρ and write the whole query as one big expression (as an alternative to using temporary relations)

$$\pi_{\text{eId, pname, dname}} \Big(\sigma_{\text{deptno = dId}} \big(\rho_{\text{eId, pname, deptno}}(\pi_{\text{empId, name, deptNo}}(\text{professor})) \times \rho_{\text{dId, dname}}(\pi_{\text{deptId, name}}(\text{department})) \big) \Big)$$

- It is easier to understand and formulate the query with *meaningfully named* temporary relations as shown earlier.
- Students are encouraged to use temporary relations.

Join Operation

- ***Cross product*** : produces all combinations of tuples
  - often only certain combinations are meaningful
  - cross product is usually followed by selection

- ***Join*** : combines tuples from two relations provided they satisfy a specified condition (join condition)
  - equivalent to performing *cross product* followed by *selection*
  - a very useful operation

- Depending on the type of condition we have
  - *theta join*
  - *equi join*

Theta join

- Let r_1 - relation with scheme $R_1 = (A_1, A_2, \ldots, A_m)$
  r_2 - relation with scheme $R_2 = (B_1, B_2, \ldots, B_n)$
  where w.l.o.g we assume $R_1 \cap R_2 = \phi$
- Notation for join expression : $r = r_1 \bowtie_\theta r_2$
- θ − the join condition - is of the form : $C_1 \wedge C_2 \wedge \ldots \wedge C_s$
  C_i is of the form : A_j <CompOp> B_k
  where <CompOp> is one of $\{ = , \neq, < , \leq , > , \geq \}$
- Scheme of the result relation r is:
  $(A_1, A_2, \ldots, A_m, B_1, B_2, \ldots, B_n)$

$$r = \{(a_1, a_2, \ldots, a_m, b_1, b_2, \ldots, b_n) \mid (a_1, a_2, \ldots, a_m) \in r_1, (b_1, b_2, \ldots, b_n) \in r_2 \text{ and } (a_1, a_2, \ldots, a_m, b_1, b_2, \ldots, b_n) \text{ satisfies } \theta\}$$

Professor

empId	name	sex	startYear	deptNo	phone
CS01	GIRIDHAR	M	1984	1	22576345
CS02	KESHAV MURTHY	M	1989	1	22576346
ES01	RAJIV GUPTHA	M	1980	2	22576244
ME01	TAHIR NAYYAR	M	1999	3	22576243

For each department, find its name and the name, sex and phone number of the head of the department

Courses

courseId	cname	credits	deptNo
CS635	Algorithms	3	1
CS636	A.I	4	1
ES456	D.S.P	3	2
ME650	Aero Dynamics	3	3

Department

deptId	name	hod	phone
1	Computer Science	CS01	22576235
2	Electrical Engg.	ES01	22576234
3	Mechanical Engg.	ME01	22576233

Example

For each department, find its name and the name, sex and phone number of the head of the department.

prof (empId, p-name, sex, deptNo, prof-phone) $\leftarrow \pi_{\text{empId, name, sex, deptNo, phone}}$ (professor)

result $\leftarrow \pi_{\text{deptId, name, hod, p-name, sex, prof-phone}}$ (department $\bowtie_{(\text{hod = empId})}$ prof)

deptId	name	hod	p-name	sex	prof-phone
1	Computer Science	CS01	Giridher	M	22576235
2	Electrical Engg.	EE01	Rajiv Guptha	M	22576234
3	Mechanical Engg.	ME01	Tahir Nayyar	M	22576233

Equi-join and *Natural join*

- *Equi-join* : Equality is the only comparison operator used in the join condition
- *Natural join* : R_1, R_2 - have common attributes, say X_1, X_2, X_3
  - Join condition:
    $(R_1.X_1 = R_2.X_1) \wedge (R_1.X_2 = R_2.X_2) \wedge (R_1.X_3 = R_2.X_3)$
    - Values of common attributes should be equal
  - Schema for the result $Q = R_1 \cup (R_2 - \{X_1, X_2, X_3\})$
    - Only one copy of the common attributes is kept

- Notation for natural join : $r = r_1 * r_2$

Example – Equi-join

Find courses offered by each department

$\pi_{\text{deptId, name, courseId, cname, credits}}$ (Department $\bowtie_{(\text{deptId = deptNo})}$ Courses)

deptId	name	courseId	cname	credits
1	Computer Science	CS635	Algorithms	3
1	Computer Science	CS636	A.I	4
2	Electrical Engg.	ES456	D.S.P	3
3	Mechanical Engg.	ME650	Aero Dynamics	3

Teaching

empId	courseId	sem	year	classRoom
CS01	CS635	1	2005	BSB361
CS02	CS636	1	2005	BSB632
ES01	ES456	2	2004	ESB650
ME01	ME650	1	2004	MSB331

To find the courses handled by each professor

Professor * Teaching

result

empId	name	sex	startYear	deptNo	phone	courseId	sem	year	classRoom
CS01	Giridhar	M	1984	1	22576345	CS635	1	2005	BSB361
CS02	Keshav Murthy	M	1989	1	22576346	CS636	1	2005	BSB632
ES01	Rajiv Guptha	M	1989	2	22576244	ES456	2	2004	ESB650
ME01	Tahir Nayyar	M	1999	3	22576243	ME650	1	2004	MSB331

Division operator

- The necessary condition to determine $r \div s$ on instances $r(R)$ and $s(S)$ is $S \subset R$
- The relation $r \div s$ is a relation on schema $R - S$.

  A tuple t is in $r \div s$ if and only if

  1) t is in $\pi_{R-S}(r)$
  2) For every tuple t_s in s, there is t_r in r satisfying both
     a) $t_r[S] = t_s$
     b) $t_r[R - S] = t$

// $t_r[S]$ − the sub-tuple of t_r consisting of values of attributes in S

- Another Definition $r = r_1 \div r_2$

Division operator produces a relation R(X) that includes all tuples t[X] that appear in r_1 in combination with every tuple from r_2 where $R_1 = Z$ and $R_2 = Y$ and $Z = X \cup Y$

R = (A, B, C, D), S = (A, B), X = (C, D)

$x = r \div s$

s

A	B
a_1	b_1
a_2	b_2

r

A	B	C	D
a_1	b_1	c_1	d_1
a_2	b_2	c_1	d_1
a_1	b_1	c_2	d_2
a_1	b_1	c_3	d_3
a_2	b_2	c_3	d_3

x

C	D
c_1	d_1
c_3	d_3

(c_2, d_2) is not present in the result of division as it does not appear in combination with all the tuples of s in r

Query using division operation

Find those students who have enrolled for *all* courses offered in the dept of Computer Science.

Step1: Get the course enrollment information for all students

studEnroll $\leftarrow \pi_{\text{rollNo, name, courseId}}$ (student * enrollment)

Step2: Get the course Ids of all courses offered by CS dept

csCourse $\leftarrow \pi_{\text{courseId}}(\sigma_{\text{dname = "Computer Science"}}(\text{courses} \bowtie_{\text{deptId = deptNo}} \text{dept}))$

Result : studEnroll $\div$ csCourse

Schema

Suppose result of step 1
(we skip roll number for simplicity)

studEnroll

name	courseId
Mahesh	CS635
Mahesh	CS636
Rajesh	CS635
Piyush	CS636
Piyush	CS635
Deepak	ES456
Lalitha	ME650
Mahesh	ME650

result of step 2

csCourse

courseId
CS635
CS636

Let's assume for a moment that student names are unique!

studEnroll $\div$ csCourse
result

name
Mahesh
Piyush

Complete Set of Operators

- Are all Relational Algebra operators essential ?
  Some operators can be realized through other operators

- What is the minimal set of operators ?
  - The operators $\{\sigma, \pi, \times, \cup, -\}$ constitute a *complete* set of operators
  - Necessary and sufficient set of operators.
  - Intersection – union and difference
  - Join – cross product followed by selection
  - Division – project, cross product and difference

Example Queries

Schema

Retrieve the list of female PhD students

$\sigma_{\text{degree} = \text{'phD'} \wedge \text{sex} = \text{'F'}}$ (student)

Obtain the name and rollNo of all female BTech students

$\pi_{\text{rollNo, name}}(\sigma_{\text{degree} = \text{'BTech'} \wedge \text{sex} = \text{'F'}}$ (student))

Obtain the rollNo of students who never obtained an 'E' grade

$\pi_{\text{rollNo}}(\sigma_{\text{grade} \neq \text{'E'}}$ (enrollment))

is incorrect!!

(what if some student gets E in one course and A in another?)

π_{rollNo} (student) – $\pi_{\text{rollNo}}(\sigma_{\text{grade} = \text{'E'}}$ (enrollment))

More Example Queries

Obtain the department Ids for departments with no lady professor

π_{deptId} (dept) – $\pi_{\text{deptId}}(\sigma_{\text{sex} = \text{'F'}}$ (professor))

Obtain the rollNo of male students who have obtained at least one S grade

$\pi_{\text{rollNo}}(\sigma_{\text{sex} = \text{'M'}}$(student)) $\cap$ $\pi_{\text{rollNo}}(\sigma_{\text{grade} = \text{'S'}}$ (enrollment))

Another Example Query

Schema

Obtain the names, roll numbers of students who have got S grade in the CS3700 course offered in 2017 odd semester along with his/her advisor name.

reqStudsRollNo $\leftarrow$
$\pi_{\text{rollNo}}(\sigma_{\text{courseId = 'CS3700' \& year = '2017' \& sem = 'odd' \& grade = 'S'}}$(enrollment))

reqStuds-Name-AdvId (rollNo, sName, advId) $\leftarrow$
$\pi_{\text{rollNo, name, advisor}}$(reqStudsRollNo * student)

result(rollNo, studentName, advisorName) $\leftarrow$
$\pi_{\text{rollNo, sName, name}}$(reqStuds-Name-AdvId $\bowtie_{\text{advId=empId}}$ professor)

Transitive Closure Queries

Schema

Obtain the courses that are either direct or indirect prerequisites of the course CS767.

- Indirect prerequisite – (prerequisite of)$^+$ a prerequisite course
- Prerequisites at all levels are to be reported

levelOnePrereq(cId1) $\leftarrow \pi_{\text{preReqCourse}}(\sigma_{\text{courseId} = \text{'CS767'}}(\text{preRequisite}))$

levelTwoPrereq(cId2) $\leftarrow$
$\pi_{\text{preReqCourse}}(\text{preRequisite} \bowtie_{\text{courseId} = \text{cId1}} \text{levelOnePrereq}))$

Similarly, level *k* prerequisites can be obtained.

But, prerequisites at all levels can not be obtained as there is no looping mechanism.

Outer Join Operation (1/2)

- Theta join, equi-join, natural join are all called *inner joins* . The result of these operations contain only the matching tuples
- The set of operations called *outer joins* are used when <u>all</u> tuples in relation *r* or relation *s* or both in *r* and *s* have to be in result.

There are 3 kinds of outer joins:

Left outer join $⟕$

Right outer join $⟖$

Full outer join $⟗$

Outer Join Operation (2/2)

Left outer join: $r ⟕ s$

It keeps all tuples in the first, or left relation *r* in the result. For some tuple *t* in *r*, if no matching tuple is found in *s* then S-attributes of *t* are made null in the result.

Right outer join: $r ⟖ s$

Same as above but tuples in the second relation are all kept in the result. If necessary, R-attributes are made null.

Full outer join: $r ⟗ s$

All the tuples in both the relations *r* and *s* are in the result.

Instance Data for Examples

Student

rollNo	name	degree	year	sex	deptNo	advisor
CS04S001	Mahesh	M.S	2004	M	1	CS01
CS05S001	Amrish	M.S	2003	M	1	null
CS04M002	Piyush	M.E	2004	M	1	CS01
ES04M001	Deepak	M.E	2004	M	2	null
ME04M001	Lalitha	M.E	2004	F	3	ME01
ME03M002	Mahesh	M.S	2003	M	3	ME01

Professor

empId	name	sex	startYear	deptNo	phone
CS01	GIRIDHAR	M	1984	1	22576345
CS02	KESHAV MURTHY	M	1989	1	22576346
ES01	RAJIV GUPTHA	M	1980	2	22576244
ME01	TAHIR NAYYAR	M	1999	3	22576243

Left outer join

$$temp \leftarrow (student ⟕_{advisor = empId} professor)$$

$$\rho_{rollNo,\ name,\ advisor} (\pi_{rollNo,\ student.name,\ professor.name} (temp))$$

Result

rollNo	name	advisor
CS04S001	Mahesh	Giridhar
CS05S001	Amrish	Null
CS04M002	Piyush	Giridhar
ES04M001	Deepak	Null
ME04M001	Lalitha	Tahir Nayyer
ME03M002	Mahesh	Tahir Nayyer

Right outer join

$$temp \leftarrow (student ⟖_{advisor = empId} professor)$$

$$\rho_{rollNo,\ name,\ advisor} (\pi_{rollNo,\ student.name,\ professor.name} (temp))$$

Result

rollNo	name	advisor
CS04S001	Mahesh	Giridhar
CS04M002	Piyush	Giridhar
null	null	Keshav Murthy
null	null	Rajiv Guptha
ME04M001	Lalitha	Tahir Nayyer
ME03M002	Mahesh	Tahir Nayyer

Full outer join

temp ← (student $⟗_{advisor = empId}$ professor)

$\rho_{\text{rollNo, name, advisor}}$ (π $_{\text{rollNo, student.name, professor.name}}$ (temp))

Result

rollNo	name	advisor
CS04S001	Mahesh	Giridhar
CS04M002	Piyush	Giridhar
CS05S001	Amrish	Null
null	null	Keshav Murthy
ES04M001	Deepak	Null
null	null	Rajiv Guptha
ME04M001	Lalitha	Tahir Nayyer
ME03M002	Mahesh	Tahir Nayyer

E/R Diagrams to Relational Schema

- E/R model and the relational model give different representations of a real world enterprise
- An E/R diagram can be converted to a collection of relations
- For each entity set and relationship set in E/R diagram we will have a corresponding relational table with the same name as entity set / relationship set
- Each table will have multiple columns whose names are obtained from the attributes of entity types/relationship types

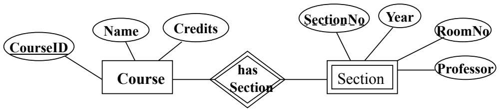
Relational representation of strong entity sets

- Create a table T_i for each strong entity set E_i.
- Include simple attributes and simple components of composite attributes of entity set E_i as attributes of T_i.
    - Multi-valued attributes of entities are dealt with separately.
- The primary key of E_i will also be the primary key of T_i.
- The primary key can be referred to by other tables via foreign keys in them to capture relationships as we see later

Relational representation of weak entity sets

- Let E' be a weak entity owned by a strong/weak entity E
- E' is converted to a table, say R', where…
- Attributes of R' will be
  - Attributes of the weak entity set E' and
    Primary key attributes of the identifying strong entity E
    (Or, partial key of E + primary key of the owner of E,
    if E is itself a weak entity)
    - These attributes will also be a foreign key in R' referring to the table corresponding to E
- Key of R' : partial key of E' + Key of E
- Multi-valued attributes of E' are dealt separately as described later

Example



Corresponding tables are

course

courseId	name	credits

section

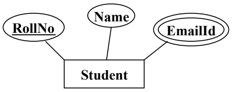
sectionNo	courseId	year	roomNo	professor

Primary key of *section* = {courseId, sectionNo}

Relational representation of multi-valued attributes

- One separate table for each multi-valued attribute
- One column for this attribute and
- Column(s) for the primary key attribute(s)
  of the table that corresponds to the entity / relationship set
  for which this is an attribute.

e.g.,



student

rollNo	name

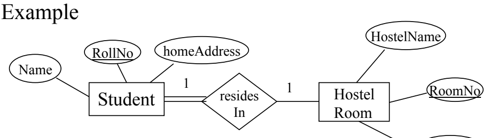
mailIds

emailId	rollNo

Handling Binary 1:1 Relationship

- Let S and T be entity sets in relationship R and S' and T' be the tables corresponding to these entity sets
- Choose an entity set which has total participation in R, if there is one (say, S)
- Include the primary key of T' as a foreign key in S' referring to relation T'
- Include all simple attributes (and simple components of composite attributes) of R as attributes of S'
- We can do the other way round too
  – lot of null values

Example



Note: Assuming every student resides in hostel.
S - Student R - residesIn T - Hostel Room

Student

<u>RollNo</u>	**Name**	**homeAddress**	**RoomId**

Hostel

<u>RoomNo</u>	**HostelName**	**address**

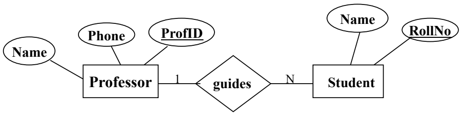
Both entity sets participate fully:
We can merge relations of both into one "merged" relation.

Foreign key name need not be same as primary key of the other relation

Handling 1: N Relationship

- Let S be the participating entity on the N-side and T the other entity. Let S' and T' be the corresponding tables.
- Include primary key of T' as foreign key in S'
- Include any simple attribute (and simple components of composite attributes) of 1:N relation type as attributes of S'

Example



Student

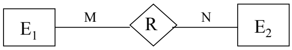
RollNo	Name	guide

Professor

ProfId	Name	phone

Handling M:N relationship

- Make a separate table T for this relationship R between entity sets E_1 and E_2.
  Let R_1 and R_2 be the tables corresponding to E_1 and E_2.
- Include primary key attributes of R_1 and R_2 as foreign keys in T. Their combination is the primary key in T.



R_1

PK_1

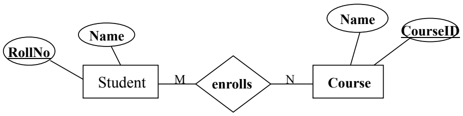
T

FK_1	FK_2

R_2

PK_2

Example



student

name	rollNo

enrollment

rollNo	courseId

course

name	courseID

Primary key of *enrollment* table is {RollNo, CourseID}

Handling Recursive relationships

- Make a table T for the participating entity set E
  (this might already be existing)
  and one table for recursive relationship R.

Example