Chapter 1

RELATIONAL DATA MODEL

The relational model of data presents a logical view of a database in which the user perceives the data to be organized in tabular form. This is a very simple and intuitive view of data which hides all the complex details of how the data is actually stored and accessed within a computer. Moreover, over the years, very sophisticated and efficient structures and algorithms have been developed to implement database systems based on the relational model. As a result, relational databases are being used overwhelmingly in the industry and anywhere else there is a need to manage large amounts of data.

In this chapter, the basic concepts and foundations of the relational data model are presented. In addition, two sample relational databases are described which will be used throughout this book.

1.1 Relational Database

The relational data model has a strong mathematical foundation based on set theory. A very brief mathematical definition of a relational database follows.

A *relation scheme* is a finite sequence of unique attribute names. For example,

```
EMPLOYEES = (EMPID, ENAME, ADDRESS, SALARY)
```

is a relation scheme with four attribute names.

A *domain* is a set of values. With each attribute name, `A`, a domain, `dom(A)`, is associated. This domain includes a special value called *null*. For example, `dom(EMPID)` could be the set of all possible integers between 1000 and 9999 and the special null value.

Given a relation scheme `R = A1, ..., An`, a *relation* `r` on the scheme `R` is defined as any finite subset of the Cartesian product

```
dom(A1) x ...  x dom(An).
```

Assuming appropriate domains for the `EMPLOYEES` relation scheme, a sample relation under this scheme could be

```
{ (1111,'Jones','111 Ash St.',20000),
  (2222,'Smith','123 Elm St.',25000),
  (3333,'Brown','234 Oak St.',30000) }
```

Each of the elements of a relation is also referred to as a *tuple*.

A *relational database scheme*, `D`, is a finite set of relation schemes,

```
{ R1, ..., Rm }.
```

A *relational database* on scheme `D` is a set of relations

```
{ r1, ...  rm }
```

where each `ri` is a relation on the corresponding scheme `Ri`.

1.2 Integrity Constraints

In addition to the data content, a relational database consists of a set of conditions, referred to as *integrity constraints*, that must be met or satisfied by the data content at all times. The relational database is referred to as a *valid* database if its data content satisfies all the integrity constraints specified in its definition. Individual relations are referred to *valid* if they satisfy all the constraints imposed on them. Three very basic and important types of constraints are discussed here.

Primary Key: A *key* for a relation scheme `R` is any subset, `K` of `R` that satisfies the property that in every valid relation under the scheme `R`, it is not possible to have two different tuples with the same values under `K`. A *candidate key* for `R` is any key for `R` such that none of its proper subsets is also a key. It is the case that every relation scheme has at least one candidate key. The *primary key* for a relation scheme `R` is one of the candidate keys chosen by the designer of the database. For the `EMPLOYEES` relation scheme, the `EMPID` attribute by itself is the primary key. It is not always the case that the primary key is a singleton. In many situations, the primary key consists of more than one attribute. The primary key attributes are required to satisfy the `not null` constraint, i.e., no tuple can have a `null` value under the primary key attributes. This property of primary keys is often referred to as the `entity integrity rule`.

Referential Integrity/Foreign Key: The referential integrity constraint is a condition that is specified across two relations. During the design of a relational database, the designer may create a relation scheme `R` which includes the primary key attributes of another relation scheme, say `S`. In such a situation, the referential integrity constraint specifies the condition that the values that appear under the primary key attributes in any valid relation under scheme `R` **must** also appear in the relation under scheme `S`. The attributes in the scheme `R` that correspond to the primary key attributes of scheme `S` collectively are referred to as a *foreign key* in scheme `R`. Unlike the primary key attributes, the foreign key attributes do not have to satisfy the `not null` constraint.

As an example, consider the `EMPLOYEES` relation scheme and the two additional relation schemes

```
PROJECTS = (PROJID,PROJNAME,LOCATION)
WORKSIN = (EID,PROJID,HOURS).
```

The `PROJECTS` relation scheme represents information about different projects and the `WORKSIN` relation scheme represents information about which employee works for which projects and

how many hours do they work. It is clear that the primary key for PROJECTS is the lone attribute PROJID. The primary key for the WORKSIN relation scheme, which represents a relationship between EMPLOYEES and PROJECTS, is the combination of EMPID and PROJID attributes. It is assumed that a single employee can work for multiple projects and that a project certainly can have many employees. The relation scheme WORKSIN includes primary key attributes from the EMPLOYEES relation scheme and the PROJECTS relation scheme. The referential integrity constraint in this situation dictates that if an EMPID value is present in a valid relation under the WORKSIN scheme then the same value must also be present in the relation under the EMPLOYEES scheme. In a similar manner, a PROJID value in a valid relation under the WORKSIN scheme must also be present in the relation under the PROJECTS scheme.

Not Null: This constraint specifies the condition that tuple values under certain attributes (specified to be not null) cannot be null. This condition is usually always imposed on the primary key attributes[1]. Other attributes may also be constrained to be not null if the need arises. In the EMPLOYEES relation scheme, the attributes EMPID and ENAME are likely candidates on which the not null constraint should be imposed.

1.3 Tabular View of a Relation

Informally, a relation as defined earlier, can also be viewed as a table made up of rows and columns. The columns are labeled with the attribute names of the relation scheme and the rows correspond to individual tuples of the relation. For example the sample relation under the EMPLOYEES relation scheme can be viewed as the following table:

[1] In Oracle, primary key attributes are automatically constrained to be `not null`.

EMPID	ENAME	ADDRESS	SALARY
1111	Jones	111 Ash St.	20000
2222	Smith	123 Elm St.	25000
3333	Brown	234 Oak St.	30000

with four columns labeled with the attribute names in the relation scheme and three rows corresponding to the three tuples in the relation. Since the relation scheme and the tuples are defined to be sequences it is important to keep the order of the components within a row in correspondence with the column names.

1.4 Sample Databases

Two databases are described in this section. The first one represents information that is usually kept in the grade books of instructors of courses in a university. The second database represents information that is usually maintained by a mail order company which sells products by mail.

Grade book database

The grade book database consists of six relations defined on the six schemes shown in Figure 1.1.

```
CATALOG(CNO,CTITLE)
STUDENTS(SID,FNAME,LNAME,MINIT)
COURSES(TERM,LINENO,CNO,A,B,C,D)
COMPONENTS(TERM,LINENO,COMPNAME,MAXPOINTS,WEIGHT)
ENROLLS(SID,TERM,LINENO)
SCORES(SID,TERM,LINENO,COMPNAME,POINTS)
```

Figure 1.1: Grade Book Database Scheme

- The `CATALOG` relation keeps information about course numbers and course titles of courses taught by a particular instructor. `CNO` is the primary key for the `CATALOG` relation.

- The STUDENTS relation keeps information about the students of a particular instructor. The SID attribute is the primary key for the STUDENTS relation.

- The COURSES relation keeps information about the various courses that have been taught by a particular instructor. The TERM attribute corresponds to the term (such as Fall97 or Spring98) in which the course was taught, the LINENO is a unique section number assigned by the registrar of the university within a term. The combination of TERM and LINENO is the primary key for this relation. The attributes A, B, C and D are numeric attributes that hold as values the scores at which the corresponding grades are assigned (for example A = 90, B = 80, C = 70, D = 60). The CNO attribute is a foreign key in this relation as it appears as a primary key in CATALOG.

- The COMPONENTS relation keeps information about the various grading components (such as homework, quizzes, exams etc.) for a particular course taught by the instructor. For each course taught, identified by the attributes TERM and LINENO, this relation records information about the grading component, the maximum points assigned to this component and the weight of this component relative to the other components. Since each course may have multiple components, the combination of the attributes TERM, LINENO and COMPNAME forms the primary key. Since the combination of TERM and LINENO appearing in this relation is a primary key for the COURSES relation, it is classified as a foreign key.

- The ENROLLS relation records information about which student was enrolled in which course taught by the instructor. The combination of all three attributes (SID, TERM and LINENO) forms the primary key. There are two foreign keys in this relation: SID referring to the STUDENTS relation and the combination of TERM and LINENO referring to the COURSES relation.

- The SCORES relation records the grading component scores (or points) for each student enrolled in a course. The combination

of the attributes SID, TERM, LINENO and COMPNAME forms the primary key for this relation. There are two foreign keys in this relation: the combination of SID, TERM and LINENO referring to the ENROLLS relation and the combination of attributes TERM, LINENO and COMPNAME referring to the COMPONENTS relation.

A sample instance of the grade book database is shown in Figure 1.2.

catalog

CNO	CTITLE
csc226	Introduction to Programming I
csc227	Introduction to Programming II
csc343	Assembly Programming
csc481	Automata and Formal Languages
csc498	Introduction to Database Systems
csc880	Deductive Databases and Logic Programming

students

SID	FNAME	LNAME	MINIT
1111	Nandita	Rajshekhar	K
2222	Sydney	Corn	A
3333	Susan	Williams	B
4444	Naveen	Rajshekhar	B
5555	Elad	Yam	G
6666	Lincoln	Herring	F

courses

TERM	LINENO	CNO	A	B	C	D
f96	1031	csc226	90	80	65	50
f96	1032	csc226	90	80	65	50
sp97	1031	csc227	90	80	65	50

Figure 1.2: Grade Book Database Instance

components

TERM	LINENO	COMPNAME	MAXPOINTS	WEIGHT
f96	1031	exam1	100	30
f96	1031	quizzes	80	20
f96	1031	final	100	50
f96	1032	programs	400	40
f96	1032	midterm	100	20
f96	1032	final	100	40
sp97	1031	paper	100	50
sp97	1031	project	100	50

enrolls

SID	TERM	LINENO
1111	f96	1031
2222	f96	1031
4444	f96	1031
1111	f96	1032
2222	f96	1032
3333	f96	1032
5555	sp97	1031
6666	sp97	1031

Figure 1.2: Grade Book Database Instance (Continued)

scores

SID	TERM	LINENO	COMPNAME	POINTS
1111	f96	1031	exam1	90
1111	f96	1031	quizzes	75
1111	f96	1031	final	95
2222	f96	1031	exam1	70
2222	f96	1031	quizzes	40
2222	f96	1031	final	82
4444	f96	1031	exam1	83
4444	f96	1031	quizzes	71
4444	f96	1031	final	74
1111	f96	1032	programs	400
1111	f96	1032	midterm	95
1111	f96	1032	final	99
2222	f96	1032	programs	340
2222	f96	1032	midterm	65
2222	f96	1032	final	95
3333	f96	1032	programs	380
3333	f96	1032	midterm	75
3333	f96	1032	final	88
5555	sp97	1031	paper	80
5555	sp97	1031	project	90
6666	sp97	1031	paper	80
6666	sp97	1031	project	85

Figure 1.2: Grade Book Database Instance (Continued)

Mail order database

The mail order database consists of six relations defined on the six schemes shown in Figure 1.3:

```
EMPLOYEES(ENO,ENAME,ZIP,HDATE)
PARTS(PNO,PNAME,QOH,PRICE,LEVEL)
CUSTOMERS(CNO,CNAME,STREET,ZIP,PHONE)
ORDERS(ONO,CNO,ENO,RECEIVED,SHIPPED)
ODETAILS(ONO,PNO,QTY)
ZIPCODES(ZIP,CITY)
```

Figure 1.3: Mail Order Database Scheme

- The `EMPLOYEES` relation keeps information about the employees of the mail order company. The `ENO` attribute is the primary key. The `ZIP` attribute is a foreign key referring to the `ZIPCODES` table.
- The `PARTS` relation keeps a record of the inventory of the company. For each part, besides its number and name, the quantity on hand, unit price and the reorder level are recorded. `PNO` is the primary key for this relation.
- The `CUSTOMERS` relation keeps information about the customers of the mail order company. Each customer is assigned a customer number, `CNO`, which serves as the primary key. The `ZIP` attribute is a foreign key referring the `ZIPCODES` relation.
- The `ORDERS` relation contains information about the orders placed by customers, the employee who took the order, the order receive and ship dates. `ONO` is the primary key. The `CNO` attribute is a foreign key referring the `CUSTOMERS` relation and the `ENO` attribute is a foreign key referring the `EMPLOYEES` table.
- The `ODETAILS` relation contains information about the various parts ordered by the customer within a particular order. The combination of `ONO` and `PNO` attributes forms the primary key.

The ONO attribute is a foreign key referring the ORDERS relation and the PNO attribute is a foreign key referring the PARTS relation.

- The ZIPCODES relation maintains information about the zip codes for various cities. ZIP is the primary key.

A sample instance of the mail order database is shown in Figure 1.4.

employees

ENO	ENAME	ZIP	HDATE
1000	Jones	67226	12-DEC-95
1001	Smith	60606	01-JAN-92
1002	Brown	50302	01-SEP-94

parts

PNO	PNAME	QOH	PRICE	LEVEL
10506	Land Before Time I	200	19.99	20
10507	Land Before Time II	156	19.99	20
10508	Land Before Time III	190	19.99	20
10509	Land Before Time IV	60	19.99	20
10601	Sleeping Beauty	300	24.99	20
10701	When Harry Met Sally	120	19.99	30
10800	Dirty Harry	140	14.99	30
10900	Dr. Zhivago	100	24.99	30

customers

CNO	CNAME	STREET	ZIP	PHONE
1111	Charles	123 Main St.	67226	316-636-5555
2222	Bertram	237 Ash Avenue	67226	316-689-5555
3333	Barbara	111 Inwood St.	60606	316-111-1234

Figure 1.4: Mail Order Database Instance

orders

ONO	CNO	ENO	RECEIVED	SHIPPED
1020	1111	1000	10-DEC-94	12-DEC-94
1021	1111	1000	12-JAN-95	15-JAN-95
1022	2222	1001	13-FEB-95	20-FEB-95
1023	3333	1000	20-JUN-97	null

odetails

ONO	PNO	QTY
1020	10506	1
1020	10507	1
1020	10508	2
1020	10509	3
1021	10601	4
1022	10601	1
1022	10701	1
1023	10800	1
1023	10900	1

zipcodes

ZIP	CITY
67226	Wichita
60606	Fort Dodge
50302	Kansas City
54444	Columbia
66002	Liberal
61111	Fort Hays

Figure 1.4: Mail Order Database Instance (Continued)

1.5 Relational Algebra

The *relational algebra* is a set of algebraic operations that take as input one (for unary operators) or two (for binary operators) relations and return as output a relation. Using these operations, one can answer arbitrary ad-hoc queries against the database content. A good understanding of the relational algebra makes the task of phrasing complex queries in SQL much easier. The relational algebraic operators are briefly introduced next and then some queries against the sample databases are answered using these operations.

The relational operators are usually classified into two categories: *set-theoretic operations* and *relation-theoretic operations*.

1.5.1 Set-theoretic operations

The set-theoretic operations include *union*, *difference*, *Cartesian product*, and *intersection*. These operations are borrowed from mathematical set theory and are applicable in the relational model because relations are nothing but sets of tuples.

The union, difference and intersection operators are binary operators that operate on two *union-compatible* relations. Two relations are union-compatible if they have the same number of attributes and the domains of the corresponding attributes in the two relations are the same. Consider two relations r and s that are union-compatible. The set-theoretic operations are defined as follows:

Union: $r \cup s = \{t | t \in r \text{ or } t \in s\}$.

In other words, the union of two union-compatible relations contains all the tuples from each of the relations.

Difference: $r - s = \{t | t \in r \text{ and } t \notin s\}$

The difference of two union-compatible relations contains all those tuples in the first relation that are not present in the second relation.

Intersection: $r \cap s = \{t | t \in r \text{ and } t \in s\}$

The intersection of two union-compatible relations contains all the tuples that are contained in both the relations.

Cartesian Product: The Cartesian product is a binary operator that takes as input two relations (r and s on any schemes) and produces a relation on the scheme that is the concatenation of the relation schemes of the input relations. The tuples in the Cartesian product are constructed by concatenating each tuple in the first input relation with each tuple in the second input relation. Formally,

$$r \times s = \{t1.t2 | t1 \in r \text{ and } t2 \in s\},$$

where, $t1.t2$ is the concatenation of tuples $t1$ and $t2$ to form a larger tuple.

Examples of the set-theoretic operations are shown in Figure 1.5.

Since the attribute names in a relation scheme must be unique, the scheme of the Cartesian product of relations r and s in the example contains attribute names prefixed by $r.$ and $s.$.

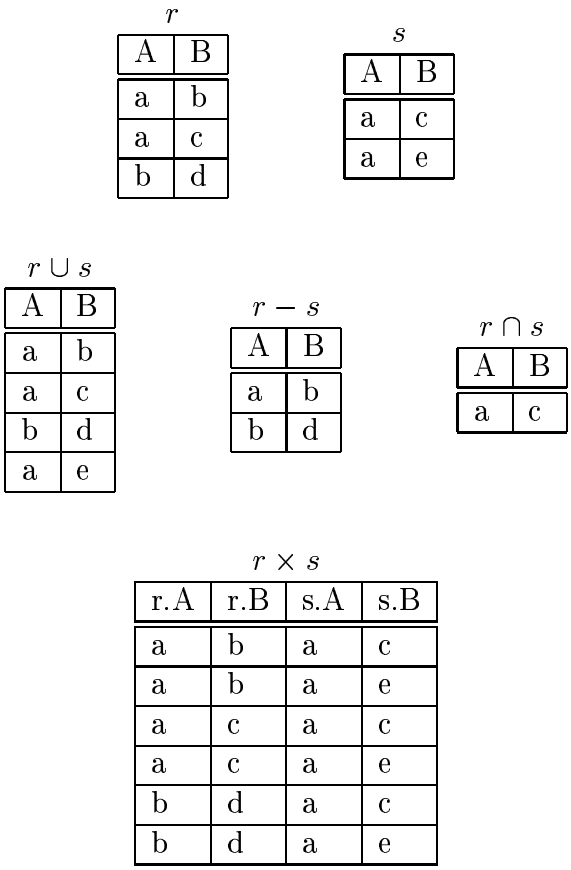
1.5.2 Relation-theoretic operations

The relation-theoretic operations include *rename*, *select*, *project*, *natural join*, and *division* among others.

Rename: The rename operator takes as input a relation and returns the same relation as output, but under a different name. This operation is useful and necessary for queries which need to refer to the same relation more than once. The symbolic notation for the rename operator is $\rho_s(r)$, where r is the input relation and s is the new name.

Select: The select operator acts as a horizontal filter for relations. Given a selection condition, the select operator produces an output relation which consists of only those tuples from the input relation that satisfy the selection condition. Symbolically, the select operator is written as $\sigma_F(r)$, where F is the selection criterion and r is the input relation. Formally, the select operator is defined as follows:

$$\sigma_F(r) = \{t | t \in r \text{ and } t \text{ satisfies } F\}.$$



r

A	B
a	b
a	c
b	d

s

A	B
a	c
a	e

$r \cup s$

A	B
a	b
a	c
b	d
a	e

$r - s$

A	B
a	b
b	d

$r \cap s$

A	B
a	c

$r \times s$

r.A	r.B	s.A	s.B
a	b	a	c
a	b	a	e
a	c	a	c
a	c	a	e
b	d	a	c
b	d	a	e

Figure 1.5: Set-theoretic Operators

Project: The project operator acts as a vertical filter for relations. Given a sub-list of attribute names of a relation, the project operator keeps only those values that correspond to the sub-list of attribute names and discards other values in tuples. Symbolically, the project operator is written as $\pi_A(r)$, where A is a sub-list of the attributes of r. Formally, the project operator is defined as follows:

$$\pi_A(r) = \{t[A] | t \in r\}$$

where $t[A]$ is a tuple constructed from t by keeping the values that correspond to the attributes in A and discarding other values.

Natural Join: The natural join operator takes as input two relations and produces as output a relation whose scheme is the concatenation of the two schemes of the input relations with any duplicate attribute names discarded. A tuple in the first input relation is said to *match* a tuple in the second input relation if they have the same values under the common attributes. The tuples in the natural join are constructed by concatenating each tuple in the first input relation with each *matching* tuple in the second input relation and discarding the values under the common attributes of the second relation. Symbolically, the natural join is written as $r \bowtie s$, where r is a relation on scheme R and s is a relation on scheme S. Formally, the natural join operation is defined as follows:

$$r \bowtie s = \{t | (\exists u \in r)(\exists v \in s)(t[R] = u \text{ and } t[S] = v)\}$$

Division: The division operator takes as input two relations, called the dividend relation (r on scheme R) and the divisor relation (s on scheme S) such that all the attributes in S also appear in R and s is not empty. The output of the division operation is a relation on the scheme R with all the attributes common with S discarded. A tuple t is put in the output of the operation if for all tuples u in s, the tuple tu is in r, where tu is a tuple constructed from t and u by combining the individual values in these tuples in the proper order to form a tuple in r. Symbolically, the division operation is written as $r \div s$ and is defined as follows:

$$r \div s = \{t | (\forall u \in s)(tu \in r)\}$$

Examples of the relation-theoretic operations are shown in Figure 1.6.

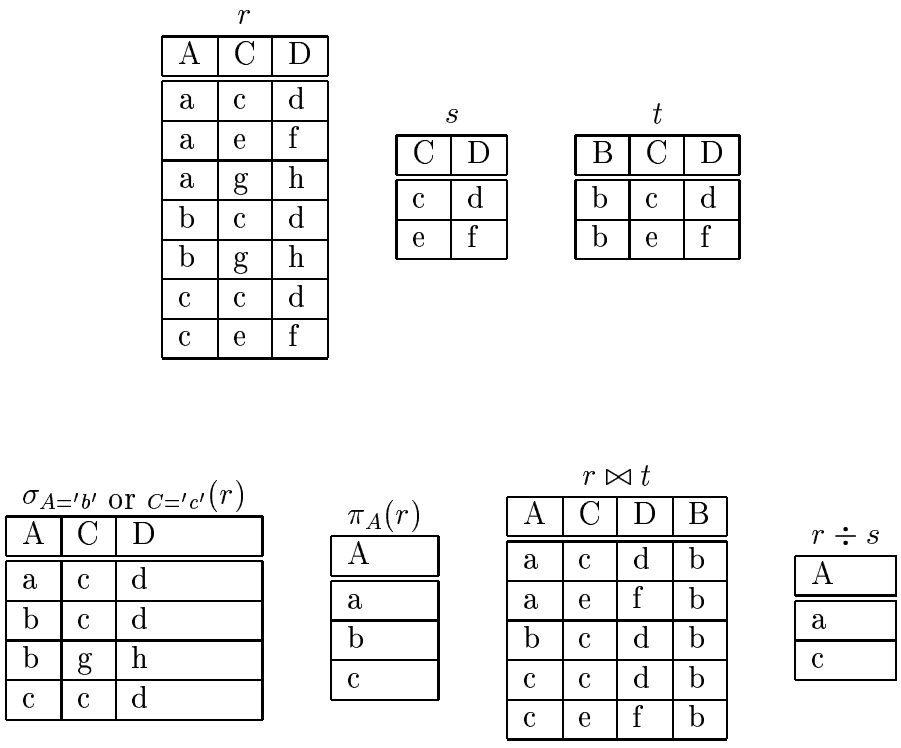


Figure 1.6: Relation-theoretic Operations

Among the relational operators presented so far, there are six basic operators: union, difference, Cartesian product, rename, select, and project. This basic set of operations has the property that none of them can be expressed in terms of the others. The remaining operators presented, namely intersection, natural join, and division can be expressed in terms of the basic operators as follows:

Intersection: $r \cap s = r - (r - s)$

Natural Join: $r \bowtie s = \pi_{R \cap S}(\sigma_F(r \times s))$
where F is a selection condition which indicates that the tuple values under the common attributes of r and s are equal.

Division: $r \div s = \pi_{R-S}(r) - \pi_{R-S}((\pi_{R-S}(r) \times s) - r)$

Even though relation schemes are defined as sequences, they are treated as sets in these equalities for simplicity.

An explanation for the equality for division is in order! First, all candidate tuples for the result are calculated by the expression

$$\pi_{R-S}(r)$$

Next, these candidate tuples are combined with all tuples of s in the following expression

$$\pi_{R-S}(r) \times s$$

to give a relation containing all combinations of candidate tuples with all tuples of s. Since we are looking for tuples under the scheme $R - S$ which combine with all tuples of s and are also present in r, if we subtract r from the previous expression, we will get all the combinations of tuples that are "missing" in r. By projecting these tuples on $R - S$, we get all those tuples that should not go to the result in the following expression.

$$\pi_{R-S}((\pi_{R-S}(r) \times s) - r)$$

Finally, we subtract this set from the set of all candidate tuples and obtain the output relation of the division operator.

1.5.3 Querying using relational algebra

The following is a list of queries against the two sample databases and the corresponding relational algebraic expressions that return the answers to the queries. The relational algebraic expressions are broken up into smaller parts and are assigned to temporary variables. One could easily write one whole expression from these individual parts and thereby not requiring the assignment primitive.

Grade book database queries

Q1 Get the names of students enrolled in the `Assembly Programming` class in the `f96` term.

$$
\begin{aligned}
t1 &:= \sigma_{CTITLE='\text{Assembly Programming}'}(catalog) \\
t2 &:= \sigma_{TERM='f96'}(courses) \\
t3 &:= t1 \bowtie t2 \bowtie enrolls \bowtie students \\
result &:= \pi_{FNAME,LNAME,MINIT}(t3)
\end{aligned}
$$

Q2 Get the `SID` values of students who did not enroll in any class during the `f96` term.

$$\pi_{SID}(students) - \pi_{SID}(\sigma_{TERM='f96'}(enrolls))$$

Q3 Get the `SID` values of students who have enrolled in `csc226` and `csc227`,

$$
\begin{aligned}
t1 &:= \pi_{SID}(enrolls \bowtie \sigma_{CNO='csc226'}(courses)) \\
t2 &:= \pi_{SID}(enrolls \bowtie \sigma_{CNO='csc227'}(courses)) \\
result &:= t1 \cap t2
\end{aligned}
$$

Q4 Get the `SID` values of students who have enrolled in `csc226` or `csc227`,

$$
\begin{aligned}
t1 &:= \pi_{SID}(enrolls \bowtie \sigma_{CNO='csc226'}(courses)) \\
t2 &:= \pi_{SID}(enrolls \bowtie \sigma_{CNO='csc227'}(courses)) \\
result &:= t1 \cup t2
\end{aligned}
$$

Q5 Get the `SID` values of students who have enrolled in **all** the courses in the catalog.

$$\pi_{SID,CNO}(courses \bowtie enrolls) \div \pi_{CNO}(catalog)$$

Mail order database queries

Q6 Get part names of parts that cost less than `20.00`.

$$\pi_{PNAME}(\sigma_{PRICE<20.00}(parts))$$

Q7 Get pairs of CNO values of customers who have the the same zip-code.

$$t1 := \rho_{c1}(customers) \times \rho_{c2}(customers)$$
$$t2 := \sigma_{c1.ZIP=c2.ZIP \text{ and } c1.CNO<c2.CNO}(t1)$$
$$result := \pi_{c1.CNO,c2.CNO}(t2)$$

Q8 Get the names of customers who have ordered parts from employees living in `Wichita`.

$$t1 := \pi_{ENO}(employees \bowtie \sigma_{CITY='Wichita'}(zipcodes))$$
$$result := \pi_{CNAME}(customers \bowtie orders \bowtie t1)$$

Q9 Get CNO values of customers who have ordered parts only from employees living in `Wichita`.

$$t1 := \pi_{ENO}(employees \bowtie \sigma_{CITY\neq'Wichita'}(zipcodes))$$
$$result := \pi_{CNO}(orders) - \pi_{CNO}(orders \bowtie t1))$$

Q10 Get CNO values of customers who have ordered parts from all employees living in `Wichita`.

$$t1 := \pi_{ENO}(employees \bowtie \sigma_{CITY='Wichita'}(zipcodes))$$
$$result := \pi_{CNO,ENO}(orders) \div t1$$