BACHELOR OF COMPUTER APPLICATIONS

SEMESTER 3

DCA2102

DATABASE MANAGEMENT SYSTEM

Unit 9

Relational Calculus

Table of Contents

1. INTRODUCTION

Relational calculus is an alternative to relational algebra. In contrast to algebra, which is procedural, the calculus is nonprocedural, or *declarative*, in that it allows us to describe the set of answers, without being explicitabout how they should be computed. Relational calculus has had a big influence on the design of commercial query languages such as SQL and, especially, Query-by-Example (QBE).

The variant of the calculus that we present in detail is called the tuple relational calculus (TRC). Variables in TRC take on tuples as values. In another variant, called the domain relational calculus (DRC), the variables range over field values. TRC has had more of an influence on SQL, while DRC has strongly influenced QBE.

1.1 Objectives:

After learning this unit, you should be able to:

- *Explain about relational calculus.*
- *Describe about tuple relational calculus and its semantics.*
- *Explain about domain relational calculus*
- *Differentiate relational ALGEBRA and relational CALCULUS*

2. TUPLE RELATIONAL CALCULUS

A **tuple variable** is a variable that takes on tuples of a particular relation schema as values. That is, every value assigned to a given tuple variable has the same number and type of fields. A tuple relational calculus query has the form { *T* | *p(T)* } where *T* is a tuple variable and *p*(*T*) denotes a *formula* that describes *T*; we will shortly define formulas and queries rigorously. The result of this query is the set of all tuples *t* for which the formula *p*(*T*) evaluates to true with *T* = *t*. The language for writing formulas *p*(*T*) is thus at the heart of TRC and is essentially a simple subset of first-order *logic*. As a simple example, consider the following query.

For Example: Find all sailors with a rating above 7. (Sailors- S is a relation)
{ S | S ε Sailors ^ S: rating > 7 }

When this query is evaluated on an instance of the Sailors relation, the tuplevariable *S* is instantiated successively with each tuple, and the test *S.rating>7* is applied. The answer contains those instances of *S* that pass this test.

Syntax of TRC Queries

We now define these concepts formally, beginning with the notion of a formula. Let *Rel* be a relation name, *R* and *S* be tuple variables, *a* an attribute of *R*, and *b* an attribute of *S*. Let op denote an operator in the set(<, >, =, ≤, ≥). An **atomic formula** is one of the following:

- R ε Rel (ε is belongs to)
- R.a op S.b
- R.a op constant, or constant op R.a

A **formula** is recursively defined to be one of the following, where *p* and *q* are themselves formulas, and *p*(*R*) denotes a formula in which the variable*R* appears:

- any atomic formula
- $\neg p$, $p \wedge q$, $p \vee q$, or $p \Rightarrow q$
- $\exists R(p(R))$, where R is a tuple variable
- $\forall R(p(R))$, where R is a tuple variable

In the last two clauses above, the **quantifiers "For any" and "For all"** are said to **bind** the variable *R*. A variable is said to be **free** in a formula or *subformula* (a formula contained in a larger formula) if the subformula does not contain an occurrence of a quantifier that binds it.

We observe that every variable in a TRC formula appears in a subformula that is atomic, and every relation schema specifies a domain for each field; this observation ensures that each variable in a TRC formula has a well-defined domain from which values for the variable are drawn. That is, each variable has a well-defined *type*, in the programming language sense. Informally, an atomic formula *R ε Rel* gives *R* the type of tuples in *Rel*, and comparisons such as *R.a* op *S.b* and *R.a* op *constant* induce type restrictions on the field *R.a*. If a variable R does not appear in an atomic formula of the form *R ε Rel* (i.e., it appears only in atomic formulas that are comparisons), we will follow the convention that the type of R is a tuple, whose fields include all (and only) fields of R that appear in the formula.

We will not define types of variables formally, but the type of a variable should be clear in most cases, and the important point to note is that comparisons of values having different types should always fail. (In discussions of relational calculus, the simplifying assumption is often made that there is a single domain of constants and that this is the domain associated with each field of each relation.)

A **TRC query** is defined to be an expression of the form { *T* | p(T) }, where *T* is the only free variable in the formula p.

2.1 Semantics of TRC Queries

What does a TRC query mean? More precisely, what is the set of answer tuples for a given TRC query? The **answer** to a TRC query { *T* | p(T) }, as we noted earlier, is the set of all tuples *t* for which the formula *p*(*T*) evaluates to true with variable *T* assigned the tuple value *t*. To complete this definition, we must state which assignments of tuple values to the free variables in a formula make the formula evaluate to true.

A query is evaluated on a given instance of the database. Let each free variable in a formula *F* be bound to a tuple value. For the given assignment of tuples to variables, with respect

to the given database instance,*F* evaluates to (or simply 'is') true if one of the following holds:

- F is an atomic formula **R ε Rel**, and R is assigned a tuple in the instanceof relation *Rel*.
- F is a comparison *R.a* op *S.b*, *R.a* op *constant*, or *constant* op *R.a*, and the tuples assigned to *R* and *S* have field values *R.a* and *S.b* that make the comparison true.
- F is of the form ¬*p*, and *p* is not true or of the form *p* ^ *q* and both *p* and*q* are true; or of the form *p v q*, and one of them is true, or of the form (*p)q* and *q* is true whenever *p* is true.
- F is of the form For Any *R*(*p*(*R*)), and there is some assignment of tuplesto the free variables in *p*(*R*), including the variable *R* that makes the formula *p*(*R*) true.
- *F* is of the form For all R(*p*(*R*)), and there is some assignment of tuplesto the free variables in *p*(*R*) that makes the formula *p*(*R*) true no matter what tuple is assigned to *R*.

2.2 Examples of TRC Queries

We now illustrate the calculus through several examples, using the instances *B*1 of Boats, *R*2 of Reserves, and *S*3 of Sailors shown in Figures 9.1, 9.2, and 9.3. We will use parentheses, as needed, to make our formulas unambiguous. Often, a formula *p*(*R*) includes a condition *R* ε *Rel* and the meaning of the phrases *some tuple R* and *for all tuples R* is intuitive. We will use the notation For any *R* ε *Rel*(*p*(*R*)) for any *R*(*R* ε *Rel* ^ *p*(*R*)).

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Fig 9.1: An Instance *S*3 of Sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Fig 9.2: An Instance *R*2 of Reserves

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Fig 9.3: An Instance *B*1 of Boats

Similarly, we use the notation For all R ε Rel(p(R)) for all R(R ε Rel) p(R)).Find the names and ages of sailors with a rating above 7.

$$\{P \mid \exists S \in Sailors(S.rating > 7 \wedge P.name = S.sname \wedge P.age = S.age)\}$$

This query illustrates a useful convention: P is considered to be a tuple variable with exactly two fields, which are called *name* and *age*, because these are the only fields of *P* that are mentioned and *P* does not range over any of the relations in the query; that is, there is no subformula of the formP ε Relname. The result of this query is a relation with two fields, *name* and *age*. The atomic formulas P.name = S.sname and P.age = S.age give values to the fields of an answer tuple P. On instances B1, R2, and S3, the answer is the set of tuples <Lubber, 55:5>, <Andy, 25:5>, <Rusty, 35.0>, <Zorba, 16.0>, and <Horatio, 35.0>.

Find the sailor name, boat id, and reservation date for each reservation.

$$\{P \mid \exists R \in Reserves \;\; \exists S \in Sailors$$
$$(R.sid = S.sid \wedge P.bid = R.bid \wedge P.day = R.day \wedge P.sname = S.sname)\}$$

For each reserve tuple, we look for a tuple in Sailors with the same *sid*. Given a pair of such tuples, we construct an answer tuple *P* with fields *sname, bid*, and *day* by copying the corresponding fields from these two tuples. This query illustrates how we can combine values from different relations in each answer tuple. The answer to this query on instances *B*1, *R*2, and *S*3 is shown in Figure 9.4.

Find the names of sailors who have reserved boat 103.

$$\{P \mid \exists S \in Sailors \;\; \exists R \in Reserves(R.sid = S.sid \wedge R.bid = 103 \wedge P.sname = S.sname)\}$$

This query can be read as follows: "Retrieve all sailor tuples for which thereexists a tuple in Reserves, having the same value in the *sid* field, and with *bid* = 103." That is, for each sailor tuple, we look for a tuple in Reserves thatshows that this sailor has reserved boat 103. The answer tuple *P* contains just one field, *sname*.

(Q2) Find the names of sailors who have reserved a red boat.

$$\{P \mid \exists S \in Sailors \ \ \exists R \in Reserves(R.sid = S.sid \wedge P.sname = S.sname \wedge \exists B \in Boats(B.bid = R.bid \wedge B.color = 'red'))\}$$

sname	*bid*	*day*
Dustin	101	10/10/98
Dustin	102	10/10/98
Dustin	103	10/8/98
Dustin	104	10/7/98
Lubber	102	11/10/98
Lubber	103	11/6/98
Lubber	104	11/12/98
Horatio	101	9/5/98
Horatio	102	9/8/98
Horatio	103	9/8/98

Fig 9.4: Answer to Query

This query can be read as follows: "Retrieve all sailor tuples *S* for which there exist tuples *R* in Reserves and B in Boats such that S.sid = R.sid, R.bid = B.bid, and B.color = 'red'." Another way to write this query, which corresponds more closely to this reading, is as follows:

$$\{P \mid \exists S \in Sailors \ \ \exists R \in Reserves \ \ \exists B \in Boats (R.sid = S.sid \wedge B.bid = R.bid \wedge B.color = 'red' \wedge P.sname = S.sname)\}$$

(Q7) Find the names of sailors who have reserved at least two boats.

$$\{P \mid \exists S \in Sailors \ \ \exists R1 \in Reserves \ \ \exists R2 \in Reserves (S.sid = R1.sid \wedge R1.sid = R2.sid \wedge R1.bid \neq R2.bid \wedge P.sname = S.sname)\}$$

Contrast this query with the algebra version and see how much simpler the calculus version is. In part, this difference is due to the cumbersome renaming of fields in the algebra version, but the calculus version really is simpler.

(Q9) Find the names of sailors who have reserved all boats.

$$\{P \mid \exists S \in Sailors \ \ \forall B \in Boats (\exists R \in Reserves(S.sid = R.sid \wedge R.bid = B.bid \wedge P.sname = S.sname))\}$$

This query was expressed using the division operator in relational algebra. Notice how easy it is expressed in calculus. The calculus query directly reflects how we might express the query in English: "Find sailors *S* such thatfor all boats *B* there is a Reserves tuple showing that sailor *S* has reserved boat *B*."

(Q14) Find sailors who have reserved all red boats.

$$\{S \mid S \in Sailors \wedge \forall B \in Boats$$
$$(B.color = 'red' \Rightarrow (\exists R \in Reserves(S.sid = R.sid \wedge R.bid = B.bid)))\}$$

This query can be read as follows: For each candidate (sailor), if a boat is red, the sailor must have reserved it. That is, for a candidate sailor, a boat being red must imply the sailor having reserved it. Observe that since we can return an entire sailor tuple as the answer instead of just the sailor's name, we have avoided introducing a new free variable (e.g., the variable *P* in the previous example) to hold the answer values. In instances *B*1, *R*2, and *S*3, the answer contains the Sailors tuples with *sid*s 22 and 31.

We can write this query without using implication, by observing that an expression of the form *p=>q* is logically equivalent to ¬*pvq*:

$$\{S \mid S \in Sailors \wedge \forall B \in Boats$$
$$(B.color \neq 'red' \vee (\exists R \in Reserves(S.sid = R.sid \wedge R.bid = B.bid)))\}$$

This query should be read as follows: "Find sailors *S* such that for all boats *B*, either the boat is not red, or a Reserves tuple shows that sailor *S* has reserved boat *B*."

Self-Assessment Questions – 1

1. A_____is a variable that takes on tuples of a particular relation schema as values.
2. A TRC stands for_____.
3. In the atomic formula clauses, the quantifiers "For any" and "For all" aresaid to _____the variable *R*.
4. A_____query is defined to be expression of the form { *T* | p(T) },where *T* is the only free variable in the formula p.
5. A query is evaluated on a given_____of the database.

3. DOMAIN RELATIONAL CALCULUS

A **domain variable** is a variable that ranges over the values in the domainof some attribute (e.g., the variable can be assigned an integer if it appears in an attribute whose domain is the set of integers). A DRC query has the form { <x_1, x_2, ... xn> | p(<x_1, x_2, ... X_n>) }, where each x_i is either a *domain variable* or a constant and p(<x_1, x_2, ... X_n>) denotes a **DRCformula** whose only free variables are the variables among the x_i ; 1 ≤ I ≤ n. The result of this query is the set of all tuples <x_1, x_2.... x_n> for which the formula evaluates to true.

A DRC formula is defined in a manner that is very similar to the definition of a TRC formula. The main difference is that the variables are now domain variables. Let op denote an operator in the set { <, >,=, ≤, ≥, ≠ } and let X and Y be domain variables.

An **atomic formula** in DRC is one of the following:

- <x_1, x_2,..... x_n> ε Rel, where Rel is a relation with n attributes; each x_i ; 1 ≤ I ≤ n is either a variable or a constant.
- X op Y
- X op *constant*, or *constant* op X

A **formula** is recursively defined to be one of the following, where p and q are themselves formulas, and $p(X)$ denotes a formula in which the variable X appears: any atomic formula.

- any atomic formula
- $\neg p$, $p \wedge q$, $p \vee q$, or $p \Rightarrow q$
- $\exists X(p(X))$, where X is a domain variable
- $\forall X(p(X))$, where X is a domain variable

The reader is invited to compare this definition with the definition of TRC formulas and see how closely these two definitions correspond. We will not define the semantics of DRC formulas formally; this is left as an exercise forthe reader.

Examples of DRC Queries

We now illustrate DRC through several examples. The reader is invited to compare these with the TRC versions.

(Q11) Find all sailors with a rating above 7.

$$\{\langle I,N,T,A\rangle \mid \langle I,N,T,A\rangle \in Sailors \wedge T > 7\}$$

This differs from the TRC version in giving each attribute a (variable) name. The condition < I,N,T,A > ε Sailors ensures that the domain variables *I*, *N*, *T*,and *A* are restricted to be fields of the *same* tuple. In comparison with the TRC query, we can say *T* > 7 instead of *S.rating* > 7, but we must specifythe tuple < I,N, T,A> in the result, rather than just *S*.

(Q1) Find the names of sailors who have reserved boat 103.

$$\{\langle N\rangle \mid \exists I,T,A(\langle I,N,T,A\rangle \in Sailors$$
$$\wedge \exists Ir,Br,D(\langle Ir,Br,D\rangle \in Reserves \wedge Ir = I \wedge Br = 103))\}$$

Notice that only the *sname* field is retained in the answer and that only *N* isa free variable. We use the notation For any Ir, Br, D(...) as a shorthand for For any Ir(For any Br(For any D(: : :))).

Very often, all the quantified variables appear in a single relation, as in this example. An even more compact notation in this case is For any < *Ir, Br, D*> ε *Reserves.* With this notation, which we will use henceforth, the above query would be as follows:

$$\{\langle N\rangle \mid \exists I,T,A(\langle I,N,T,A\rangle \in Sailors$$
$$\wedge \exists \langle Ir,Br,D\rangle \in Reserves(Ir = I \wedge Br = 103))\}$$

The comparison with the corresponding TRC formula should now be straightforward. This query can also be written as follows; notice the repetition of variable I and the use of the constant 103:

$$\{\langle N\rangle \mid \exists I,T,A(\langle I,N,T,A\rangle \in Sailors$$
$$\wedge \exists D(\langle I,103,D\rangle \in Reserves))\}$$

(Q2) Find the names of sailors who have reserved a red boat.

$$\{\langle N\rangle \mid \exists I,T,A(\langle I,N,T,A\rangle \in Sailors$$
$$\wedge \exists \langle I,Br,D\rangle \in Reserves \wedge \exists \langle Br,BN,'red'\rangle \in Boats)\}$$

(Q7) Find the names of sailors who have reserved at least two boats

$$\langle\rangle$$
$$\langle\rangle \mid \{N \mid \exists I, T, A$$
$$(I,N,T,A \in Sailors \wedge$$
$$\exists Br1, Br2, D1, D2 ($$
$$I, Br1, D1 \in Reserves \wedge$$
$$I, Br2, D2 \in Reserves \wedge Br1 \neq Br2)$$

⟨⟩⟨⟩Notice how the repeated use of variable I ensures that the same sailor has reserved both the boats in question.

(Q9) Find the names of sailors who have reserved all boats.

$$\{\langle N\rangle \mid \exists I,T,A(\langle I,N,T,A\rangle \in Sailors \wedge$$
$$\forall B,BN,C(\neg(\langle B,BN,C\rangle \in Boats) \vee$$
$$(\exists\langle Ir,Br,D\rangle \in Reserves(I = Ir \wedge Br = B))))\}$$

This query can be read as follows: "Find all values of *N* such that there is some tuple <I,N,T,A> in Sailors satisfying the following condition: for every <B, BN, C >, either this is not a tuple in Boats or there is some tuple <Ir, Br,D> in Reserves that proves that Sailor I has reserved boat *B*." The For all quantifier allows the domain variables *B*, *BN*, and *C* to range overall values in their respective attribute domains, and the pattern ' ¬(<*B, BN,C*> ε *Boats*) ν ' is necessary to restrict attention to those values that appear in tuples of Boats. This pattern is common in DRC formulas, and the notation For all <*B, BN,C*> ε *Boats* can be used as a shorthand instead. This is similar to the notation introduced earlier for *for any*. With this notationthe query would be written as follows:

$$\{\langle N\rangle \mid \exists I,T,A(\langle I,N,T,A\rangle \in Sailors \wedge \forall\langle B,BN,C\rangle \in Boats$$
$$(\exists\langle Ir,Br,D\rangle \in Reserves(I = Ir \wedge Br = B)))\}$$

(Q14) Find sailors who have reserved all red boats.

$$\{\langle I,N,T,A\rangle \mid \langle I,N,T,A\rangle \in Sailors \wedge \forall\langle B,BN,C\rangle \in Boats$$
$$(C = 'red' \Rightarrow \exists\langle Ir,Br,D\rangle \in Reserves(I = Ir \wedge Br = B))\}$$

Here, we find all sailors such that for every red boat there is a tuple inReserves that shows the sailor has reserved it.

Self-Assessment Questions – 2

6. A_____is a variable that ranges over the values in the domainof some attribute.
7. A DRC formula is defined in a manner that is very similar to thedefinition of a _____.
8. The main difference between the DRC and TRC is that the variablesare now _____variables.

4. RELATIONAL ALGEBRA VS RELATIONAL CALCULUS

Relational algebra is a procedural language while relational calculus is a non- procedural language. We have presented two formal query languages forthe relational model. Are they equivalent in power? Can every query thatcan be expressed in relational algebra also be expressed in relational calculus? The answer is yes, it can. Can every query that can be expressed in relational calculus also be expressed in relational algebra? Before we answer this question, we consider a major problem with calculus as we have presented it.

Consider the query { S | ¬(S ε Sailors) }. This query is syntactically correct. However, it asks for all tuples *S* such that *S* is not in (the given instance of) Sailors. The set of such *S* tuples is obviously infinite, in the context of infinitedomains such as the set of all integers. This simple example illustrates an *unsafe* query. It is desirable to restrict relational calculus to disallow unsafe queries.

We now sketch how calculus queries are restricted to be safe. Consider a set I of relation instances, with one instance per relation that appears in the query *Q*. Let Dom(Q,I) be the set of all constants that appear in these relation instances I, or in the formulation of the query *Q* itself. Since we only allow finite instances I, Dom(Q, I) is also finite.

For a calculus formula *Q* to be considered safe, at a minimum we want to ensure that for any given I, the set of answers for *Q* contains only values that are in *Dom*(*Q, I*).

While this restriction is obviously required, it is not enough. Not only do we want the set of answers to be composed of constants in *Dom*(*Q,I*), we wish to *compute* the set of answers by only examining tuples that contain constants in *Dom*(*Q, I*)! This wish leads to a subtle point associated with theuse of quantifiers For all and For any: Given a TRC formula of the form For any *R*(*p*(*R*)), we want to find all values for variable *R,* that make this formulatrue by checking only tuples that contain constants in *Dom*(*Q, I*). Similarly, given a TRC formula of the form For all *R*(*p*(*R*)), we want to find any values for variable *R,* that make this formula false, by checking only tuples that contain constants in *Dom*(*Q, I*).

We therefore define a *safe* TRC formula *Q* to be a formula such that:

1. For any given I, the set of answers for *Q* contains only values that are in Dom(Q,I).
2. For each subexpression of the form For any R(p(R)) in *Q*, if a tuple *r* (assigned to variable *R*) makes the formula true, then *r* contains only constants in *Dom*(*Q, I*).
3. For each subexpression of the form For all *R*(*p*(*R*)) in *Q*, if a tuple *r* (assigned to variable *R*) contains a constant that is not in *Dom*(*Q, I*), then *r* must make the formula true.

Note that this definition is not *constructive*, that is, it does not tell us how to check if a query is safe.

The query *Q* = { *S* |¬(*S 2 Sailors*) } is unsafe by this definition. Dom(Q,I) is the set of all values that appear in (an instance *I* of) Sailors. Consider the instance *S*1 shown in Figure 9.1. The answer to this query obviously includes values that do not appear in *Dom*(*Q, S*1). Returning to the questionof expressiveness, we can show that every query that can be expressed using a *safe* relational calculus query, can also be expressed as a relational algebra query. The expressive power of relational algebra is often used as ametric of how powerful a relational database query language is. If a query language can express all the queries that we can express in relational algebra, it is said to be **relationally complete**. A practical query language isexpected to be relationally complete; in addition, commercial query languages typically support features that allow us to express some queries that cannot be expressed in relational algebra.

Self-Assessment Questions – 3

9. Can every query that can be expressed in relational algebra also beexpressed in relational calculus?
10. If a query language can express all the queries that we can express inrelational algebra, it is said to be_____.

5. SUMMARY

In this unit, we dealt with that instead of describing a query by how to compute the output relation, a *relational calculus* query describes the tuples in the output relation. The language for specifying the output tuples is essentially arestricted subset of first-order predicate logic. In *tuple relational calculus*, variables take on tuple values and in *domain relational calculus*, variables take on field values, but the two versions of the calculus are very similar.

All relational algebra queries can be expressed in relational calculus. If we restrict ourselves to safe queries on the calculus, the converse also holds. An important criterion for commercial query languages is that they should be*relationally complete* in the sense that they can express all relational algebra queries.

6. TERMINAL QUESTION

1. What is *relational completeness*? If a query language is relationallycomplete, can you write any desired query in that language?
2. What is an *unsafe* query? Give an example and explain why it isimportant to disallow such queries.
3. Let the following relation schemas be given:

   $R = (A,B,C)$

   $S = (D,E, F)$

   Let relations $r(R)$ and $s(S)$ be given. Give an expression in the tuplerelational calculus that is equivalent to each of the following:

   a. $\Pi_A(r)$
   b. $\sigma_{B=17}(r)$
   c. $r \times s$
   d. $\Pi_{A,F}(\sigma_{C=D}(r \times s))$

   Let $R = (A, B, C)$, and let $r1$ and $r2$ both be relations on schema R. Give an expression in the domain relational calculus that is equivalent to eachof the following:

   a. $\Pi_A(r1)$
   b. $\sigma_{B=17}(r1)$
   c. $r1 \cup r2$
   d. $r1 \cap r2$
   e. $r1 - r2$
   f. $\Pi_{A,B}(r1) \cup_{B,C} (r2)$
4. How do you differentiate relational algebra and relational calculus?

7. ANSWERS

Self-Assessment Questions

1. tuple variable
2. tuple relational calculus
3. bind
4. TRC
5. Instance
6. domain variable
7. TRC formula
8. Domain
9. Yes
10. relationally complete

Terminal Questions

1. If a query language can express all the queries that we can express in relational algebra, it is said to be relationally complete. Yes, we can write the desired query in that language if features are supported. (Refer section 4)
2. Queries where the set of *S* tuples is obviously infinite in the context of infinite domains such as the set of all integers then such queries are unsafe queries.
3. Refer the whole unit for detail.
4. All relational algebra queries can be expressed in relational calculus. If we restrict ourselves to safe queries on the calculus, the converse also holds.