Unit IV
Relational Database Manipulation

Relational Database Manipulation

Database manipulation means the retrieval of information stored in the database, the insertion of new information into the database, and the deletion of information from the database. This unit, however, discusses only that portion which involves information retrieval.

To enable users to access or manipulate data in a database, there is that database component called data manipulation language (DML). There are basically two types of DML, namely procedural and non-procedural. The difference between the two types is that procedural DMLs require you to specify what data are needed and how to get them. Non-procedural DMLs only require you to specify the data needed.

Non-procedural DMLs are usually easier to learn and use than procedural DMLs. However, since a user does not have to specify how to get the data, these languages may generate codes which are not as efficient as those produced by procedural languages.

Relational Algebra is a procedural language which consists of a set of operations that take one or two relations as input and produce a new relation as their result. In Module 9, we will look into these different relational algebra operators and perform different operations.

Module 10 discusses a more user-friendly non-procedural language, SQL. SQL uses some relational algebra constructs. Although SQL is referred to as a query language, it contains many other capabilities besides querying a database. It includes features for defining the structure of data, for modifying data in the database, and for specifying security constraints.

Module 9
Relational Algebra

Although the concept is similar, relational algebra is very different from the algebra we learned in high school. In high school algebra, variables represent numbers. Operations like addition, subtraction, multiplication, and division operate on numeric quantities. For relational algebra, however, the variables are relations, and the operations manipulate relations to form new relations. For example, the '+' operator or union combines the tuples of one relation with the tuples of another relation. The result is a third relation.

Objectives

At the end of this module, you should be able to:

1. List the different relational algebra operators; and
2. Perform the different relational algebra operations.

It has not been formally stated previously, but we can consider relations as sets. The tuples of a relation can be considered elements of a set. Therefore, the operations that can be performed on sets can also be performed on relations. We will go over these set operations one by one and discuss other operators particular to relational algebra.

Relational Algebra

Relational algebra is a collection of operations that are used to manipulate entire relations. These operations are used, for example, to combine related tuples from several relations for the purpose of specifying an information retrieval request on a database. The result of each operation is a relation which can be further manipulated.

The relational operations are usually divided into two groups. One group includes set operations from mathematical set theory. Set operations include Union, Set Difference, Intersection, and Product. The other group consists of operations developed specifically for relational databases. These include the operations Project, Select and Join, among others. I will discuss the set operations first, because these are the fundamental operations. Then the discussion will proceed to the Project and Select operations and finally the Join operation.

Set Operations from Mathematical Set Theory

Union

The **union** of two relations R and S, denoted as R ∪ S, is formed by combining the tuples from relation R with those of relation S to produce a third. Thus, the resulting relation contains the same number of attributes as R and S and the set of all tuples t belonging to either A or B or possibly both. Duplicate tuples are eliminated. For this operation to work, the relations must be **union compatible**. This means that each relation must have the same number of attributes, and the attributes in corresponding columns must come from the same domain.

Example:

Relations A and B below are union compatible. The tuples in relation A correspond to students whose surname (LAST) starts with the letter 'A.' The tuples in relation B, on the other hand, are honor students. Then A ∪ B is the relation whose tuples include either honor students or students with last names beginning with the letter A or both. Notice that the result has three tuples, not four (since duplicate tuples were eliminated).

A

ID	Last	Grade
123	Ada	1.25
789	Aho	2.54

B

ID	Last	Grade
123	Ada	1.25
456	Clark	1.18

A ∪ B

123	Ada	1.25
789	Aho	2.54
456	Clark	1.18

SAQ 9-1

Now, it is your turn to perform the union operation on these two relations. Write your answer in the space below. When you are done, see ASAQ 9-1 for the answer.

STD_PROF

SName	PName
Susan	John
Ricardo	John
Amy	Smith

PROF_ADV

PName	AName
John	Ricardo
Smith	Susan
John	Amy

Still easy, huh? Go on to the next topic and see if it poses more challenge to you.

Set difference

The **difference** between two union compatible relations R and S, denoted as **R - S** (read as R minus S), is the set of tuples in R but not in S. The resulting relation will have the same number of attributes as R and S. As with arithmetic, the order of subtraction matters. R - S is not the same as S - R.

Example 1:

Given the relations R and S, the set difference is written below. Notice that the tuples (a, b, c) and (c, b, d) are indeed instances of R and not of S.

R

A	B	C
a	b	c
d	a	f
c	b	d

S

D	E	F
b	g	a
d	a	f

R - S

a	b	c
c	b	d

Example 2:

Study the STUDENT relation and the INSTRUCTOR relation below. Are the relations union compatible? Yes, they are. Both have two columns each. The values for the first and second columns come from the same domain – that of names.

STUDENT

FName	LName
Susan	Yao
Barbara	Jones
Amy	Ford
Francis	Johnson
Ernest	Gilbert

INSTRUCTOR

FName	LName
John	Smith
Richard	Brown
Susan	Yao
Francis	Johnson

STUDENT - INSTRUCTOR

FName	LName
Barbara	Jones
Amy	Ford
Ernest	Gilbert

INSTRUCTOR - STUDENT

FName	LName
John	Smith
Richard	Brown

Notice that STUDENT - INSTRUCTOR is different from INSTRUCTOR - STUDENT. Again, in set difference the order of the operands is important.

SAQ 9-2

Given the JUNIOR relation and the HONOR relation, what is HONOR - JUNIOR? Write your answer on the space provided. See if you can answer this before I can say "pneumonoultramicroscopicsilicovolcanocaniosis" which happens to be the longest word in the English dictionary (the last time I checked).

JUNIOR

ID	Name	Major
123	Jones	History
158	Parks	Math
271	Smith	Biology

HONOR

ID	Name	Interest
105	Andres	Zoology
123	Jones	History

Intersection

The **intersection** of two union compatible relations R and S, denoted as **R ∩ S**, is the set of all tuples t belonging to both R and S. The resulting relation will have the same number of attributes as relations R and S.

Example 1:

Again, using the relations R and S, the intersection of these relations is shown below. Clearly, only the instance (d, a, f) belongs to both relations R and S.

R

A	B	C
a	b	c
d	a	f
c	b	d

S

D	E	F
b	g	a
d	a	f

R ∩ S

d	a	f

Example 2:

Using the same relations in Example 1, the intersection of the relations STUDENT and INSTRUCTOR is given below. The instance (Susan, Yao) and (Francis, Johnson) can be found existing in both the operands/relations.

STUDENT

FName	Lname
Susan	Yao
Barbara	Jones
Amy	Ford
Francis	Johnson
Ernest	Gilbert

INSTRUCTOR

FName	LName
John	Smith
Richard	Brown
Susan	Yao
Francis	Johnson

STUDENT ∩ INSTRUCTOR

Susan	Yao
Francis	Johnson

SAQ 9-3

Given the JUNIOR relation and the HONOR relation, what is JUNIOR ∩ HONOR?

JUNIOR

ID	Name	Major
123	Jones	History
158	Parks	Math
271	Smith	Biology

HONOR

ID	Name	Interest
105	Andres	Zoology
123	Jones	History

Product

The **product** of two relations R and S (sometimes called the **Cartesian product**), denoted as **R x S**, is a concatenation of every tuple of relation R with every tuple of relation S. The product of relation R (having m tuples and x attributes) and relation S (having n tuples and y attributes) has **m** times **n** tuples and **x** plus **y** attributes. R and S do not have to be union compatible.

Example 1:

If we have the following relations R (having 3 tuples and 3 attributes) and S (having 2 tuples and 3 attributes), then R x S will have 6 tuples and 6 attributes as shown below.

R

A	B	C
a	b	c
d	a	f
c	b	d

S

D	E	F
b	g	a
d	a	f

R x S

A	B	C	D	E	F
a	b	c	b	g	a
a	b	c	d	a	f
d	a	f	b	g	a
d	a	f	d	a	f
c	b	d	b	g	a
c	b	d	d	a	f

Example 2:

Let us consider a new set of relations. What if you were working in a dating service and were given a list of names of boys (BOY) and a list of names of girls (GIRL). You are tasked to make possible pairs out of these names. Then all you need to do is get the product BOY x GIRL.

BOY

BName
Jack
Ted
Hansel

GIRL

GName
Jill
Nancy
Gretel

BOY x GIRL

Bname	GName
Jack	Jill
Jack	Nancy
Jack	Gretel
Ted	Jill
Ted	Nancy
Ted	Gretel
Hansel	Jill
Hansel	Nancy
Hansel	Gretel

But we all know that pairing people is not as simple as this. There is a great possibility that some of these pairs will not work out because of one reason or another (i.e., incompatible personalities). So let's leave that problem to the psychologists.

What you can do is set some criteria in selecting the "best" set of pairings. Actually, the product of two relations is most commonly used by another kind of operation (JOIN) which we will discuss later.

In the meantime, have your pen ready and try to do the SAQ.

SAQ 9-4

Given the STUDENT relation and the ENROLLMENT relation below, what is STUDENT x ENROLLMENT? A space below is provided for your answer.

STUDENT

Student#	Name	Major	Classification	Age
123	Jones	History	Junior	20
105	Parks	Math	Graduate	27
158	Anderson	Management	Freshman	18
225	Smith	History	Junior	22

ENROLLMENT

Student#	Subject
123	SOSN 111
105	AMAT 201
123	HIST 180

Operations for Relational Databases

Projection

Projection is an operation that selects specified attributes from a relation. The result of the projection is a new relation having the selected attribute. Simply put, **projection picks columns out of a relation**. Formally, if **R** is a relation with k attributes, we let pi, i_2, …, im **(R)**, where the ij's are distinct integers in the range 1 to k, denote the projection of R onto components i_1, i_2, …, im.

Example:

Here are two sample projections on the relation R.

R

A	B	C
a	b	c
d	a	f
c	b	d

$\pi_{A,C}(R)$

A	C
a	c
d	f
c	d

$\pi_{B}(R)$

B
b
a
b

SAQ 9-5

For the projections defined below, refer to the STUDENT relation in SAQ 9-4. Write the results of these projections on the space provided.

a) $\pi_{\text{Name, Major}}$ (STUDENT)

b) $\pi_{\text{Major, Classification}}$ (STUDENT)

Selection

Whereas the projection operator takes a vertical subset (columns) of a relation, the **selection** operator *takes the horizontal subset (rows)*. Projection identifies attributes to be included in the new relation. Selection identifies tuples to be included in the new relation. Selection makes use of a formula that serves as a guide on how to identify the tuples to be included in the new relation. First, let's define what a formula is.

Let F be a formula involving the following:

1. operands that are constants or component numbers,
2. the arithmetic comparison operators $<$, $>$, $\leq$, $\geq$, $=$, $\neq$, and
3. the logical operators $\wedge$ (and), $\vee$ (or), and $\neg$ (not).

Formally now, given a relation R, a selection is denoted as $\sigma_F(R)$, and is the set of tuples t in R such that when, for all i, we substitute the ith component of t for any occurrences of the number i in formula F, the formula F becomes true.

It sounds complicated but once you see the examples, you'll learn that things are a lot easier than they seem.

Example 1:

Here are some examples of selections on the relation R. The first selection takes all rows in R that has a value 'b' in Column B. The other selection is a little bit more complicated. It selects all rows in R such that the value in Column A is not 'a' and the value in Column C is not 'c'.

R

A	B	C
a	b	c
d	a	f
c	b	d

$\sigma_{B=b}(R)$

A	B	C
a	b	c
c	b	d

$\sigma_{(A\neq a)\wedge(C\neq c)}(R)$

A	B	C
d	a	f
c	b	d

Example 2:

Assume there exists a relation named EMPLOYEE that contains all data (name, SSS#, birthday, address, sex, salary and the department where he belongs) on employees of an organization. The relation shown below is the set of tuples that resulted from the selection:
$\sigma_{(Dept\#=4 \wedge Salary>25000) \vee (Dept\#^{1}5 \wedge Salary>30000)}(EMPLOYEE)$.

FName	MI	LName	SSS#	BDay	Address	Sex	Salary	Dept#
Ben	J	Franks	88395	08/12/45	638 Aurora Ave., Manila	M	40000	5
Ellen	L	Borg	47444	20/06/31	291 Laya St., Quezon City	F	43000	4
Howard	F	Slater	58611	15/09/52	975 Mortdale, Laguna	M	38000	5

SAQ 9-6

Again, refer to the STUDENT relation in SAQ 9-4. What are the results of the following selections? Carefully look at the values in the STUDENT relation and make sure that the tuples you include in the new relation satisfy the formula. Write your answers in the space provided.

a) $\sigma_{Age \leq 25}$ (STUDENT)

b) $\sigma_{Major='BIOLOGY'}$ (STUDENT)

Natural join

The join operation comes in several different varieties. The most important is the **natural join** (or simply **join**), and is represented by the symbol Ω□. Natural join is computed as follows:

Given two relations R and S,
1. compute R x S
2. for each attribute A that names both a column in R and a column in S, select from R x S those tuples whose values agree in the columns for R.A and S.A
3. for each attribute A above, project out the column S.A

From the given steps above, we can say that the join is an operation that combines product, selection, and (possibly) projection operations. As a note, R.A means that attribute A belongs to relation R and similarly S.A denotes the attribute A of relation S.

Example 1:

Below are the relations R, S and R · S.

R

A	B	C
a	b	c
d	b	c
b	b	f
c	a	d

S

B	C	D
b	c	d
b	c	e
a	d	b

R · S

A	B	C	D
a	b	c	d
a	b	c	e
d	b	c	d
d	b	c	e
c	a	d	b

Example 2:

Let's go back to the scenario in the example under Projection where BOY meets GIRL in a dating service. Of course, a dating service will gather other information about their clients. For the purpose of our discussion, consider that aside from the client's name, the client's favorite color is also known and that this will be the basis for our matching. We are given

BOY

BName	Color
Jack	Red
Ted	Blue
Hansel	Green

GIRL

GName	Color
Jill	Red
Nancy	Blue
Gretel	Green

Notice that to form the join BOY · GIRL, we first take the product of BOY and GIRL. This operation produces the relation below.

BOY x GIRL

BName	Color	GName	Color
Jack	Red	Jill	Red
Ted	Blue	Jill	Red
Hansel	Green	Jill	Red
Jack	Red	Nancy	Blue
Ted	Blue	Nancy	Blue
Hansel	Green	Nancy	Blue
Jack	Red	Gretel	Green
Ted	Blue	Gretel	Green
Hansel	Green	Gretel	Green

Now, we select those tuples from the product where (favorite) Color of BOY equals (favorite) Color of GIRL. This operation results in a relation with two identical attributes. One of these two is unnecessary, so one is removed by projection. Thus, the resulting relation is the one shown below.

BOY · GIRL

BName	GName	Color
Jack	Jill	Red
Ted	Nancy	Blue
Hansel	Gretel	Green

Simple! Now try your hand on the SAQ.

SAQ 9-7

Once again, refer to the STUDENT and ENROLLMENT relations in SAQ 9-4. If you perform the natural join operation on these two relations, what would the new relation look like? Write the new relation below.

Summary

Relational algebra consists of a group of relational operators that can be used to manipulate relations to obtain a desired result. In this module, I introduced the different relational algebra operations and showed some examples on how these operations work. Fundamental operations like union, set difference, intersection require that the relation operands be union compatible.

Projection and selection operations were created specifically for relational databases. Simply put, projection allows us to choose the columns to be included in a relation while selection allows us to choose the rows.

Natural join is an operation that makes use of three fundamental operations namely, product, selection and projection. Normally, unless otherwise specified, when we say join, we mean the natural join.

That wraps it up for this module. I smell something burning in the kitchen. Are you cooking something? You might as well eat your breakfast/lunch/dinner as you have successfully finished this part of the course. 'Til our next topic. Bye!

References

Adamski, J., and P. Pratt. 1991. *Database systems: management and design.* Boston, Massachusetts: Boyd & Fraser Publishing Co.

Date, C. J. 1990. *An introduction to database systems*. Reading, Massachusetts: Addison-Wesley Publishing Co., Inc. 5th ed.

Elmasri, R., and S. B. Navathe. 1994. *Fundamentals of database systems.* Redwood City, CA: The Benjamin/Cummings Publishing Co., Inc.

Silberschatz, A., H. F. Korth, and S. Sudarshan. 1997. *Database system concepts*. Singapore: The McGraw-Hill Companies, Inc. International ed.

Ullman, J. D. 1982, *Principles of database systems*. Rockville, Maryland: Computer Science Press, Inc.

Answers to Self-Assessment Questions

ASAQ 9-1

The union of relations STD_PROF and PROF_ADV is

STD_PROF ∪ PROF_ADV

Susan	John
Ricardo	John
Amy	Smith
John	Ricardo
Smith	Susan
John	Amy

Relations STD_PROF and PROF_ADV are union compatible. Both relations have two attributes each and all of these attributes come from a single domain, let's say a domain of first names. The number of tuples equals the sum of the tuples of STD_PROF and PROF_ADV. Why is this? Simply because no two tuples are the same, hence no duplicate tuples to eliminate.

ASAQ 9-2

The correct answer is

ID	Name	Interest
105	Andres	Zoology

If you got the following relation as your answer

ID	Name	Major
158	Parks	Math
271	Smith	History

then you committed a mistake. The latter relation is the set difference of JUNIOR and HONOR, where HONOR was subtracted from JUNIOR (JUNIOR - HONOR). Remember, the order of the operands is important.

ASAQ 9-3

Don't get cross-eyed now. ☺ The answer should be the relation

JUNIOR ∩ HONOR

ID	Name	Interest
123	Jones	History

Do you think Mr. Jones can answer the question, "when did Christopher Columbus discover America?"

Seriously now, of the 3 tuples of JUNIOR and of the two tuples of HONOR, only the instance (123, Jones, History) is both in JUNIOR and HONOR. Hence, the intersection of these two relations is the tuple shown above.

ASAQ 9-4

Each tuple of STUDENT is matched with every tuple of ENROLLMENT. Four tuples of STUDENT times three tuples of ENROLLMENT will therefore give the resulting relation 12 tuples. The relation should look like the table below.

STUDENT x ENROLLMENT

Student#	Name	Major	Classification	Age	Student#	Subject
123	Jones	History	Junior	20	123	SOSN 111
123	Jones	History	Junior	20	105	AMAT 201
123	Jones	History	Junior	20	123	HIST 180
105	Parks	Math	Graduate	27	123	SOSN 111
105	Parks	Math	Graduate	27	105	AMAT 201
105	Parks	Math	Graduate	27	123	HIST 180
158	Anderson	Management	Freshman	18	123	SOSN 111
158	Anderson	Management	Freshman	18	105	AMAT 201
158	Anderson	Management	Freshman	18	123	HIST 180
225	Smith	History	Junior	22	123	SOSN 111
225	Smith	History	Junior	22	105	AMAT 201
225	Smith	History	Junior	22	123	HIST 180

ASAQ 9-5

Simply picking up the chosen attributes, the resulting relations should be

a) $\pi_{\text{Name, Major}}$ (STUDENT)

Name	Major
Jones	History
Parks	Math
Anderson	Management
Smith	History

b) $\pi_{\text{Major, Classification}}$ (STUDENT)

Major	Classification
History	Junior
Math	Graduate
Management	Freshman

Notice that the resulting relation to the letter b) projection contains only three tuples as opposed to four. Actually, the instance (History, Junior) occurred twice in the new relation after the projection. Since a property of a relation is that it cannot contain duplicate tuples, to satisfy this property only one instance of the (History, Junior) tuple was retained.

ASAQ 9-6

The new relation resulting from the selection $\sigma_{\text{Age}\leq 25}$ (STUDENT) is as follows:

Student#	Name	Major	Classification	Age
123	Jones	History	Junior	20
158	Anderson	Management	Freshman	18
225	Smith	History	Junior	22

Verifying, we see that all the students in this relation are aged 25 and below.

The second question is a trick question. The selection operation will return no tuple. That is, no tuple in the STUDENT relation has as a value 'BIOLOGY' in the Major column or simply no student majors biology. Hence, the selection will return NULL.

ASAQ 9-7

WOW! Another feather in your cap! Looks like manipulating relations is not that hard, huh? But before we dwell on that, let us compare answers. The new relation should look like this

Name	Major	Classification	Student#	Age	Subject
Jones	History	Junior	123	20	SOSN 111
Parks	Math	Graduate	105	27	AMAT 201
Jones	History	Junior	123	20	HIST 180

The STUDENT and ENROLLMENT relations have Student# as a common attribute. Following the steps in computing the join of two relations, we first get the product of STUDENT and ENROLLMENT. Since Student# is a common attribute, then STUDENT.Student# must agree with ENROLLMENT.Student# or the value of STUDENT.Student# must be equal to the value of ENROLLMENT.Student#. Finally, we project out STUDENT.Student#.

Module 10
SQL Queries

A database management system requires a query language to enable users to access data. This is where SQL comes in. SQL is an acronym for Structured Query Language. By far, SQL is the best known and most commonly used language by commercial relational database systems.

Objectives

After studying this module, you should be able to:

1. Write SQL queries to retrieve data from a relational database; and
2. Predict the outcome of a valid SQL select statement.

Originally, SQL was called SEQUEL (Structured English QUEry Language). It was designed and implemented by IBM Research as the interface for an experimental relational database system called SYSTEM R in the mid-1970's. Now, the American National Standards Institute (ANSI) and the International Standards Organization (ISO) have adopted SQL as the standard language for manipulating relational database systems.

You may have noticed that SQL is an English-like language. You are correct. It uses words such as create, drop, select, update as part of its command set. I am not attempting to provide a comprehensive discussion of SQL but merely to illustrate its major features. Actually, the discussion will only concentrate on data retrieval. Join me now as I present to you queries in SQL.

A Review of Relational Databases

Before I start discussing SQL queries, let me refresh you with some relational database concepts. A relational database is one which is perceived as being composed of a collection of relations (or tables), where each relation can be thought of as a file of records of the same type. These relations, in turn, are simply a collection of rows of data (or records), where each row in a relation contains a set of values for the same group of attributes, or columns. The tables below show examples of relations in a relational database.

Example:

The EMPLOYEE relation has six attributes, namely Employee number (Emp#), Employee Name (EName), Position, Manager, Salary and Department Number (Dept#). Its primary key is Emp#. Notice that there is a NULL value present in the Manager column. This implies that King has no manager.

The next relation, DEPARTMENT, has three columns and contains four tuples. Its primary key is Dept# and it exists as a foreign key in the EMPLOYEE relation. This implies that a relationship exists between these two relations. We can say that a certain employee, whose description is contained in the EMPLOYEE relation belongs to a department described by the DEPARTMENT relation.

EMPLOYEE

Emp#	Ename	Position	Manager	Salary	Dept#
7369	SMITH	CLERK	7902	800	20
7499	ALLEN	SALESMAN	7698	1600	30
7566	JACK	MANAGER	7839	2975	20
7654	MARTIN	SALESMAN	7698	1250	30
7698	LIN	MANAGER	7839	2850	30
7782	ACE	MANAGER	7839	2450	10
7839	KING	PRESIDENT		5000	10
7844	TURNER	SALESMAN	7698	1500	30
7876	ADAMS	CLERK	7788	1100	20
7900	JAMES	CLERK	7698	950	30
7902	FORD	ANALYST	7566	3000	20
7934	MILLER	CLERK	7782	1300	10

DEPARTMENT

Dept#	DName	Location
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Study these relations very well. We will refer to these examples frequently for the rest of the module.

Basic SQL Queries

The SQL language has four basic data manipulation statement types, namely SELECT, INSERT, UPDATE and DELETE. The last three constructs alter the data in tables. On the other hand, the SELECT statement retrieves information from the database, implementing all the operators of Relational Algebra. The basic form of the SELECT statement, sometimes called the **basic query block**, is composed of the clauses SELECT and FROM and has the following syntax:

```
SELECT    <attribute list>
FROM      <table list>;
```

where <attribute list> is a list of attribute names whose values are to be retrieved by the query; and, <table list> is a list of the relation names required to process the query.

Example 1:

To list all the names of the employees and the number of the department where they belong in the EMPLOYEE table, the corresponding query is:

```
SELECT    Dept#, Ename
FROM      Employee;
```

The above statement will return the following result:

Dept#	Ename
20	SMITH
30	ALLEN
20	JACK
30	MARTIN
30	LIN
10	ACE
10	KING
30	TURNER
20	ADAMS
30	JAMES
20	FORD
10	MILLER

Note that when more than one attribute is included in the attribute list, the attributes are separated by a comma. The table above is actually taken from the EMPLOYEE relation where the other columns Emp#, Position, Manager and Salary were projected out. Notice that the presentation of columns corresponds to the sequence in which the attributes were listed in the SELECT statement and not as ordered in the original relation.

Example 2:

To select all columns from the table, an * (asterisk) can be specified to replace the attribute list. Thus, the select statement below will result in the same set of tuples in the DEPARTMENT relation.

```
SELECT *
FROM   Department;
```

Dept#	DName	Location
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

The DISTINCT Clause

Unless otherwise stated, SQL displays the results of a query without eliminating duplicate entries. To display only one row to represent all rows that have identical column values, we include the DISTINCT qualifier in the SELECT command.

Example 2:

With the statement SELECT Dept# FROM EMPLOYEE; the result is

Dept#
20
30
20
30
30
10
10
30
20
30
20
10

To eliminate duplicate values in the result, we include the DISTINCT clause in the SELECT command and issue the statement:

```
SELECT    DISTINCT Dept#
FROM      Employee;
```

This statement will produce the desired result shown below.

Dept#
10
20
30

The ORDER BY Clause

Usually, the order of rows returned in a query result is not defined. The ORDER BY clause may be used to specify the order in which the retrieved data will be displayed or written. If used, ORDER BY must always be the last clause in the SELECT statement. Note that the default sort ordering is ASCending, that is:

- numeric values lowest first
- date values earliest first
- character values alphabetically.

Example:

The corresponding command to list the names of all the employees with the highest salary to the lowest is:

```
SELECT      EName, Salary
FROM        Employee
ORDER BY    Salary Desc;
```

and will result in:

EName	Salary
KING	5000
FORD	3000
JACK	2975
LIN	2850
ACE	2450
ALLEN	1600
TURNER	1500
MILLER	1300
MARTIN	1250
ADAMS	1100
JAMES	950
SMITH	800

SAQ 10-1

First, get a whole sheet of paper. Next, consider the relation below.

Model#	Model	Maker	Capacity
M1	Corolla	Toyota	1.6
M2	Corolla	Toyota	2.0
M3	Crown	Toyota	2.0
M4	Accord	Honda	1.8
M5	Prelude	Honda	2.0
M6	Fairlane	Ford	2.6
M7	Charade	Daihatsu	1.4

SAQ 10-1 continued

Below are two queries written in English whose answers can be lifted from the CAR relation above. Write the corresponding SELECT statement for each query and draw the resulting table.

1. What are the names of the different car manufacturers (makers) that are included in the CAR table?

2. List the model, the manufacturer and the engine capacity of all the cars starting with the car model with the lowest engine capacity?

The WHERE Clause

The WHERE clause corresponds to the **selection** operation of Relational Algebra. It contains a condition which rows must meet in order to be included in the result. It compares values in columns, literal values or arithmetic expressions. The WHERE clause expects three elements: a column name, a comparison operator and a column name, constant or list of values.

Comparison operators are used on the WHERE clause and can be divided into two categories, namely logical and SQL.

Logical operators

These logical operators will test the following conditions:

Operator	Meaning
=	equal to
>	greater than
> =	greater than or equal to
<	less than
< =	less than or equal to

Character strings and dates in the WHERE clause must be enclosed in single quotation marks. Character strings must match case with the column value.

Example:

To list the names of all managers from the EMPLOYEE table, we write:

```
SELECT    EName, Position
FROM      Employee
WHERE     Position = 'MANAGER';
```

and the corresponding result is the table below.

EName	Position
JACK	MANAGER
LIN	MANAGER
ACE	MANAGER

SQL Operators

There are four SQL operators which operate with all data types:

Operator	Meaning
BETWEEN x AND y	between two values x and y (inclusive)
IN (<list>)	match any of a list of values
LIKE	match a character pattern
IS NULL	is a null value

Let's look at each more closely.

The BETWEEN operator

The BETWEEN operator tests for values between, and inclusive of, a low and high range.

Example:

If I want to display the names of all employees who receive a salary between 1000 and 2000, the corresponding SELECT statement is

```
SELECT    EName
FROM      Employee
WHERE     Salary BETWEEN 1000 AND 2000;
```

and this gives the corresponding table shown below.

EName	Salary
ALLEN	1600
MARTIN	1250
TURNER	1500
ADAMS	1100
MILLER	1300

The IN operator

The IN operator tests for the membership of values in a set of values. The set of values to be compared is specified in the search condition as a list of values.

Example:

In this example, we are retrieving all information on employees that belong to departments 10 and 30. The corresponding SELECT statement is

```
SELECT    *
FROM      Employee
WHERE     Dept# IN (10, 30);
```

and the resulting table produced is shown below.

Emp#	EName	Position	Manager	Salary	Dept#
7499	ALLEN	SALESMAN	7698	1600	30
7654	MARTIN	SALESMAN	7698	1250	30
7698	LIN	MANAGER	7839	2850	30
7782	ACE	MANAGER	7839	2450	10
7839	KING	PRESIDENT	null	5000	10
7844	TURNER	SALESMAN	7698	1500	30
7900	JAMES	CLERK	7698	950	30
7934	MILLER	CLERK	7782	1300	10

Note that if character, character string or date values are used in the list of values, they must be enclosed in single quotes (e.g., WHERE Position IN ('MANAGER', 'CLERK')).

The LIKE operator

The **LIKE** operator allows you to specify a search condition that searches character data for a specific pattern. This is particularly useful when you do not know the exact value to search for. Using the LIKE operator it is possible to select rows that match a character pattern. The character pattern matching operation may be referred to as a "wild card" search. Two symbols can be used to construct the search string.

Symbol	**Represents**
%	any sequence of zero or more characters
_	any single character

Example 1:

To list the names and salary of all employees whose name starts with the letter 'A', we query:

```
SELECT    EName, Position
FROM      Employee
WHERE     EName LIKE 'A%';
```

and this will give you the result:

EName	Salary
ALLEN	1600
ACE	2450
ADAMS	1100

Example 2:

What if you are looking for someone whose name you can't recall but only remember that his name has five letters in them? Well, you can choose among the rows returned by the query

```
SELECT      EName
FROM        Employee
WHERE       EName LIKE '_ _ _ _ _';
```

Now, who among these employees are you looking for?

EName
SMITH
ALLEN
ADAMS
JAMES

Note that the % (percent) and _ (underscore) may be used in any combination with literal characters. Also, note that the set of underscores in the WHERE clause in Example 10.10 is separated by spaces. This was done to emphasize that there are five and only five '_'s to correspond to the five letters. In forming actual SQL queries, the spaces are not necessary.

The IS NULL operator

The **IS NULL** operator specifically tests for values that are NULL. In the EMPLOYEE relation, only one cell (under the Manager column) has no value or we can say contains a NULL value. A query corresponding to this situation is given by the example below.

Example:

Are there employees who have no manager? The corresponding SELECT statement is:

```
SELECT    EName, Manager
FROM      Employee
WHERE     Manager IS NULL;
```

And obviously, the result is:

EName	Manager
KING	null

Negating Expressions

The following operators are negative tests:

Operator	**Meaning**
< >	not equal to
NOT <col_name> =	not equal to
NOT <col_name> >	not greater than

Operator	**Description**
NOT BETWEEN	not between two given values
NOT IN (<list>)	not in given list of values
NOT LIKE	not like string
IS NOT NULL	is not a null value

Example 1:

To find information on all employees whose salary is not between a range, we write:

```
SELECT      *
FROM        Employee
WHERE       Salary NOT BETWEEN 2000 AND 4000;
```

and the expected outcome is:

Emp#	EName	Position	Manager	Salary	Dept#
7369	SMITH	CLERK	7902	800	20
7499	ALLEN	SALESMAN	7698	1600	30
7654	MARTIN	SALESMAN	7698	1250	30
7839	KING	PRESIDENT		5000	10
7844	TURNER	SALESMAN	7698	1500	30
7876	ADAMS	CLERK	7788	1100	20
7900	JAMES	CLERK	7698	950	30
7934	MILLER	CLERK	7782	1300	10

Example 2:

To find all other department locations except Dallas, we write:

```
SELECT      Location
FROM        Department
WHERE       NOT Location = 'DALLAS';
```

This gives you the table:

Location
NEW YORK
CHICAGO
BOSTON

The WHERE clause of the SELECT statement can be replaced by:

```
WHERE       Location < > 'DALLAS';
            or
WHERE       Location NOT IN ('DALLAS');
```

and the result will still be the table shown above.

Querying Data with Multiple Conditions

The AND and OR operators may be used to make compound logical expressions. The AND predicate will expect both conditions to be 'true'; whereas the OR predicate will expect either condition to be 'true.' Thus, a simple expression can be made into a more complicated one by using either one of these two connectors.

In the following two examples, the conditions are the same. The only difference is that the first SELECT statement uses AND while the second uses OR. See how the result is dramatically changed.

Example 1:

If we are looking for the names, positions and salaries of all employees in Department 30 who earn between 1000 and 2000, we use the SELECT statement below.

```
SELECT    EName, Position, Salary
FROM      Employee
WHERE     Salary BETWEEN 1000 AND 2000
and       Dept# = 30;
```

This query will yield the table:

EName	Position	Salary
ALLEN	SALESMAN	1600
MARTIN	SALESMAN	1250
TURNER	SALESMAN	1500

Example 2:

If we change the AND to an OR in the WHERE clause in the previous SELECT statement, we expect results that are different from the one shown above. By simply replacing AND, we changed the meaning of the query. It now means: display the names, positions and salaries of all employees who either work in the department with Dept# 30 or who earn between 1000 and 2000. The equivalent SELECT statement is:

```
SELECT    EName, Position, Salary
FROM      Employee
WHERE     Salary BETWEEN 1000 AND 2000
or        Dept# = 30;
```

and the resulting table is of course different as you can see below.

EName	Position	Salary
ALLEN	SALESMAN	1600
MARTIN	SALESMAN	1250
LIN	MANAGER	2850
TURNER	SALESMAN	1500
ADAMS	CLERK	1100
JAMES	CLERK	950
MILLER	CLERK	1300

Operator Precedence

All operators are arranged in a hierarchy that determines their precedence. In an expression, operations are performed in order of their precedence, from highest to lowest. Below is a list of operators in order of their precedence. All comparison and SQL operators are performed first and the OR operator last. Where operators of equal precedence are used next to each other, they are performed from left to right.

1. All of the comparison and SQL operators have equal precedence: =, <, >, <=, >=, < >, BETWEEN…AND…, IN, LIKE, IS NULL.
2. NOT (to reverse a logical expression's result)
3. AND
4. OR

Whenever you are in doubt about which of the two operations will be performed first when an expression is evaluated, feel free to use parentheses to clarify your meaning and ensure that it is performed as what you had intended.

Example:

Suppose you want to find all the clerks in any department and all the salesmen in Department 30 only. Then the query you need to submit should be:

```
SELECT    EName, Position, Dept#
FROM      Employee
WHERE     Position = 'CLERK'
or        ( Position = 'SALESMAN'
and       Dept# = 30);
```

Note that the English query used the conjunction 'and' to mean 'plus,' while the SQL query used the operator OR. Also, the above parentheses are not necessary since AND has a higher precedence than OR, but they clarify the meaning of the expression.

SAQ 10-2

Refer to the CAR relation in SAQ 10-1. Now, write the corresponding SELECT statement for each of the queries below and draw the resulting table. Use the space provided for your answers. I listed my answers at the end of this module.

1. I prefer cars made by Toyota, Honda or Daihatsu and I go for economical rather than powerful engines. A car with an engine capacity between 1.4 and 1.8 is just about right. Can you come up with a list of cars that suits my taste?

2. A friend of mine told me that this particular car model (which I cannot recall but I think starts with the letter 'C') with capacity not more than 2.0 is a good buy. He also suggested that anything manufactured by Honda is great. Translate this query to an appropriate SELECT statement.

Joins

A **join** is used when an SQL query requires data from more than one table on the database. Rows in one table may be joined to rows in another according to common values existing in corresponding columns. There are many types of join conditions, but the most common is the **equi-join** which is similar to the natural join in Relational Algebra.

Example:

In order to ascertain, manually, which department an employee is in, we would compare the value in the EMPLOYEE's Dept# column with the same Dept# values in the DEPARTMENT relation. The relationship between EMPLOYEE and DEPARTMENT is an equi-join, in that the values in the Dept# column on both relations are equal.

To join the EMPLOYEE relation and the DEPARTMENT relation, we can issue the SELECT statement:

```
SELECT      Department.Dept#, EName, Position, DName
FROM        Employee, Department
WHERE       Employee.Dept# = Department.Dept#
ORDER BY    Department.Dept#;
```

The resulting table is as follows:

Dept#	EName	Position	DName
10	ACE	MANAGER	ACCOUNTING
10	KING	PRESIDENT	ACCOUNTING
10	MILLER	CLERK	ACCOUNTING
20	SMITH	CLERK	RESEARCH
20	JACK	MANAGER	RESEARCH
20	ADAMS	CLERK	RESEARCH
20	FORD	ANALYST	RESEARCH
30	ALLEN	SALESMAN	SALES
30	MARTIN	SALESMAN	SALES
30	LIN	MANAGER	SALES
30	TURNER	SALESMAN	SALES
30	JAMES	CLERK	SALES

Notice that every employee now has a respective department name displayed. The rows of EMPLOYEE are combined with the rows of DEPARTMENT and the rows are only displayed if the value of EMPLOYEE.Dept# and DEPARTMENT.Dept# are equal.

Observe also that the join condition specifies the names of the relations prior to the column name. This is a requirement when column names are the same in both relations.

SAQ 10-3

Refer to the CAR relation in SAQ 10-1 and consider the OWNER relation below.

OWNER

OR#	Name	Sex	Address	Model#	Color
OR00312	Famela	F	21 Apple Valley, Northampton	M3	Blue
OR00479	Albert	M	2 Parkside Lane, Creston	M2	Black
OR00101	Francis	M	16 Hosley Drive, Bourbon	M2	Gray
OR00521	Elmo	M	1A Abbot Apt., Sureney	M1	Green
OR00672	Yayeen	F	63 Driftwood Road, Cornville	M4	Red
OR00289	Richard	M	41 Honor Street, Bellbond	M3	Green
OR00165	Alex	M	A411 Armstrong, Gleeville	M2	Silver
OR00678	Pinky	F	3/39 Morrah Street, Sierra	M4	Red
OR00410	Mario	M	20 Fullerton Parkway, Eastwing	M6	Green
OR00981	Lou	F	348 Oakhill, Turquoise	M4	Blue

Now, write the corresponding SELECT statement appropriate for the situation below and draw the resulting table. Use the space provided for your answers.

If I were a car salesman and would like to know the kinds of cars male customers prefer, how can I use the CAR and OWNER relations to come up with the information that I need?

Summary

In this module, I presented the basic SQL query SELECT statement. The following clauses have been covered:

SELECT	[DISTINCT] <column list>
FROM	<table list>
WHERE	<condition(s)> / <join condition>
ORDER BY	<column/expression> [ASC/DESC];

SELECT	select at least one column from specified table(s).
*	may be used to represent all columns.
DISTINCT	can be used to eliminate duplicates.
FROM <table>	de3notes the table(s) where the columns originate.
WHERE	restricts query to rows that meet a non-join condition. It may contain column values, expressions and literals. It may also contain a join condiiton to specify how two tables will be joined together.
AND/OR	may be used in a WHERE clause to construct more complex conditions. The connector AND takes priority over OR.
()	can be used to force priority.
ORDER BY	always appears last. Specifies the sort order. One or more columns may be specified.
ASC	ascending order is the default sorting order and need not be specified.
DESC	reverses the default sorting order and must be specified after the column name.

Another day, another module. How about a dip in the pool? Ithink this module has exercised your mind quite well. Don't you think it's a good idea to exercise other parts of your body as well? So pack your gym bag and ask a friend to accompany you to the nearest pool. See you in the next module.

References

Adamski, J. and P. Pratt. 1991. *Database systems: management and design*. Boston, Massachusetts: Boyd & Fraser Publishing Co.

Date, CJ. 1990. *An introduction to database systems*. Reading, Massachusetts: Addison-Wesley Publishing Co., Inc. 5th ed.

Silberschatz, A., HF. Korth, and S. Sudarshan. 1997. *Database system concepts*. Singapore: The McGraw-Hill Companies, Inc. International ed.

Answers to Self-Assessment Questions

ASAQ 10-1

Okay now. The answer to the first query is:

```
SELECT    DISTINCT Maker
FROM      Car;
```

The resulting table is:

Maker
Daihatsu
Ford
Honda
Toyota

If you got it correctly, here's a 👍. If you drew a similar table to the one shown above, only that the order of the rows is changed, then you still get a 👍. Remember, one of the properties of a relation is that the order of the rows is not significant.

How about the next query? Your answer should be:

```
SELECT    Model, Maker, Capacity
FROM      Car
ORDER BY  Capacity;
```

and the resulting table is:

Model	Maker	Capacity
Charade	Daihatsu	1.4
Corolla	Toyota	1.6
Accord	Honda	1.8
Corolla	Toyota	2.0
Crown	Toyota	2.0
Prelude	Honda	2.0
Fairlane	Ford	2.6

You need not write the qualifier ASC after the attribute Capacity in the ORDER BY clause. But if you did, no problem. It is still correct. If you got this one correct too, then you get 👍👍. It was not that hard, was it?

ASAQ 10-2

The SELECT statement equivalent to the first query is:

```
SELECT      *
FROM        Car
WHERE       Maker IN ('Toyota', 'Honda', 'Daihatsu')
and         Capacity BETWEEN 1.4 and 1.8;
```

With this SELECT statement, I will then be able to choose any of the rows in the table:

Model#	Model	Maker	Capacity
M1	Corolla	Toyota	1.6
M4	Accord	Honda	1.8
M7	Charade	Daihatsu	1.4

The next query is a little trickier than the first. You should have come up with:

```
SELECT      *
FROM        Car
WHERE       ( Model LIKE 'C%'
and         Capacity < 2.0)
or          Maker = 'Honda';
```

The expected result of this query is found in the table below.

Model#	Model	Maker	Capacity
M1	Corolla	Toyota	1.6
M4	Accord	Honda	1.8
M5	Prelude	Honda	2.0
M7	Charade	Daihatsu	1.4

It must be pointed out that there might be other ways of representing the above queries using SQL. Rely on the discussions and your best judgment to assess if your own version of the query is also correct. Shall I give you 👍👍 or 👎👎?

By the way, thanks for the help. Now I can pick which car to buy. If only I can find the money. ☹

ASAQ 10-3

The information that are useful are contained in the table below.

Name	Model#	Color	Model	Maker	Capacity
Albert	M2	Black	Corolla	Toyota	2.0
Francis	M2	Gray	Corolla	Toyota	2.0
Elmo	M1	Green	Corolla	Toyota	1.6
Richard	M3	Green	Crown	Toyota	2.0
Alex	M2	Silver	Corolla	Toyota	2.0
Mario	M6	Green	Fairlane	Ford	2.6

In this case, we need to select the details of the car models owned by males. Therefore the SELECT statement must:

- list the displayed columns above;
- use a WHERE clause containing the comparison operator '=' to reduce the set of values to only male owners; and,
- use the connector AND to add a join condition to the WHERE clause to establish the relationship between rows of the CAR and OWNER relations.

Actually, the table above is the result of a query similar to the one below.

```
SELECT    Name, Owner.Model#, Color, Maker, Capacity
FROM      Owner, Car
WHERE     Sex = 'M'
and       Owner.Model# = Car.Model#
```

Again, I must point out that there might be other ways of representing the above query. You could have chosen to project more or lesser columns than the one I included here. Or you could opt to project Car.Model# rather than project Owner.Model# which I did above.

There you have it. The answer to our last SAQ for this module. If you got this one correctly, then you have a chance to win (drum roll)… a BRAND NEW CAR!

Seriously now, you can actually drill yourself with many more exercises. Simply think of scenarios wherein retrieval of information from a given relation is needed. Then try writing the equivalent SQL query for it. The more you practice, the easier it will be for you to create reports using SQL.