TABLE 5.2 SQL summary of fundamental EMPLOYEE TRAINING queries.

Query	√SQL
Q_σ	`select * from employee E where E.eSalary > 100000;`
Q_π	`select distinct E.eLast, E.eFirst, E.eTitle from employee E;`

$Q_\cup$	`select E.eID from employee E where E.eTitle='Manager'`
	`union`
	`select E.eID from employee E where E.eTitle='Coach';`

Q_-	`select E.eID from employee E where E.eTitle='Manager'`
	`except`
	`select T.eID from takes T;`

$Q_\times$	`select E.eID, C.cID from employee E, trainingCourse C ;`

TABLE 5.3 SQL summary of additional EMPLOYEE TRAINING queries.

Query	√SQL

$Q_\cap$	`select E.eID from employee E where E.eTitle='Manager'`
	`intersect`
	`select T.eID from takes T;`

$Q_{\bowtie_\theta}$	`select *`
	`from employee E, technologyArea A`
	`where E.eID=A.aLeadID;`

$Q_\bowtie$	`select distinct C.cTitle, T.tYear, T.tMonth, T.tDay`
	`from trainingCourse C, takes T`
	`where C.cID=T.cID ;`

Difference. The difference example query over the EMPLOYEE TRAINING enterprise retrieves the identification number of employees who are managers that have *not* taken any training courses. The previous version of the query used the except operator in SQL after defining the `managers` and `takenCourse` intermediate tables:

```
select * from managers except select * from takenCourse;
```

This query can also be specified in one step in SQL using an `exists` condition in the `where` clause:

✓
```
select E.eID
from   employee E
where  E.eTitle='Manager' and not exists
       (select  *
       from     takes T
       where    T.eID=E.eID);
```

Division. The division example query over the abstract domain finds the a's from the `abTable` that are related to *all* of the b's specified in the `bTable`, where the schema of the tables are `abTable(a,b)` and `bTable(b)`. Since SQL does not provide for the specification of universal quantification, SQL's specification of division uses the logically equivalent definition in terms of existential quantification:

✓
```
select distinct T.a
from   abTable T
where  not exists
       (select  *                                    -- β
       from     bTable B
       where    not exists
                (select  *                           -- αβ
                from     abTable AB
                where    AB.a=T.a and AB.b=B.b));
```

employees in the database:

```
√A1:   select  min(E.eSalary), max(E.eSalary),
               avg(E.eSalary), sum(E.eSalary), count(*)
       from    employee E;
```

This query returns a single tuple with five attribute values. The `min`, `max`, `avg` and `sum` aggregate functions apply to a numeric attribute. The `count(*)` counts the number of tuples in the result of the `from` clause, since there is no `where` clause specified.

As another example, consider a query that counts the number of employees that took a training course in the database technology area:

```
√A2:   select  count(distinct T.eID)
       from    dbCourse D, takes T
       where   D.cID = T.cID;
```

Grouping. In the previous example, the aggregation applied to all of the tuples in the result of the (`from` and) `where` clause. Some queries require applying an aggregate function to groups of tuples that have the same value on a grouping of attributes. SQL provides a `group by` clause for the specification of the grouping attributes. For example, consider the query that finds the minimum, maximum, and average salary of employees by title:

```
√G1:   select    E.eTitle, min(E.eSalary),
                 max(E.eSalary), avg(E.eSalary)
       from      employee E
       group by  E.eTitle;
```

Count how many same values in a column:

Relation Name	# Tuples
top10CDs	11
top10CDs_tmp1	11
top10CDs_tmp2	5
final_question2	5

cdCode/char	groupCode/char
'C1'	'G1'
'C2'	'G2'
'C3'	'G2'
'C4'	'G3'
'C5'	'G3'
'C6'	'G4'
'C7'	'G4'
'C8'	'G4'
'C9'	'G4'
'C10'	'G5'
'C12'	'G5'

=>

Relation Name	# Tuples
top10CDs	11
top10CDs_tmp1	11
top10CDs_tmp2	5
final_question2	5

groupCode/char	numberOfTop10CDs/numeric
'G5'	2
'G4'	4
'G3'	2
'G2'	2
'G1'	1

top10CDs_tmp2(groupCode,numberOfTop10CDs):=select groupCode, count(*) from top10CDs_tmp1 group by groupCode;

Max, Min, Avg Cal for one column but group by another column:

Relation Name	# Tuples
cds2000	15
cds2000_tmp1	70
totalTracks	15
totalTracks_tmp1	15
final_question3	4

cdCode/char	tracks/numeric	year/numeric
'C1'	8	2001
'C4'	3	2003
'C5'	3	2003
'C7'	3	2003
'C8'	10	2003
'C9'	9	2002
'C10'	2	2003
'C11'	2	2003
'C12'	1	2001
'C13'	7	2000
'C14'	5	2000
'C18'	3	2003
'C19'	3	2001
'C20'	3	2003
'C21'	8	2001

=>

Relation Name	# Tuples
cds2000	15
cds2000_tmp1	70
totalTracks	15
totalTracks_tmp1	15
final_question3	4

year/numeric	maxNumber/numeric	minNumber/numeric	avgNumber/numeric
2000	7	5	6
2001	8	1	5
2002	9	9	9
2003	10	2	3.625

%Cal year, maxNumber, minNumber, avgNumber

final_question3(year, maxNumber, minNumber, avgNumber):=select year, max(tracks), min(tracks),avg(tracks) from totalTracks_tmp1 group by year order by year;

Division

Relation Name	# Tuples
clients_detroit	3
info	17
gCode	2
final_question4	2

groupCode/char	clientID/char
'G1'	'Client1'
'G2'	'Client6'
'G2'	'Client4'
'G3'	'Client1'
'G3'	'Client3'
'G4'	'Client3'
'G4'	'Client1'
'G4'	'Client2'
'G5'	'Client1'
'G2'	'Client5'
'G5'	'Client5'
'G1'	'Client4'
'G5'	'Client6'
'G6'	'Client4'
'G6'	'Client5'
'G6'	'Client6'
'G6'	'Client2'

/

Relation Name	# Tuples
clients_detroit	3
info	17
gCode	2
final_question4	2

clientID/char
'Client4'
'Client5'
'Client6'

=>

Relation Name	# Tuples
clients_detroit	3
info	17
gCode	2
final_question4	2

groupCode/char
'G2'
'G6'

%cal info/clients_detroit the groupCode such that every client in Detroit rented at least one of their CDs.

```
gCode:=select distinct T.groupCode
from info T
where not exists
    (select *
     from clients_detroit B
     where not exists
          (select *
            from info AB
            where AB.groupCode = T.groupCode and AB.clientID=B.clientID));
```