SELECT Statement: The GROUP BY Clause

10.1 INTRODUCTION

The GROUP BY clause groups rows on the basis of similarities among them. For example, we could group all the rows in the PLAYERS table on the basis of the place of residence. The result would be one group of players per town. From there, we could query the number of players in each group. The final result answers the question, how many players live in each town? Other examples are: How many matches has each team played, and how much has each player incurred in penalties? In short, the GROUP BY clause is frequently used to formulate questions based on the word *per*.

By adding aggregation functions, such as COUNT and SUM, to a select block with the use of a GROUP BY clause, data can be *aggregated*. These functions owe their name to this. Aggregation means that we ask for summations, averages, frequencies, and subtotals instead of individual values.

DEFINITION

10.2 GROUPING ON ONE COLUMN

The simplest form of the GROUP BY clause is the one in which only one column is grouped. Previous chapters gave several examples of statements with such a GROUP BY clause. For the sake of clarity, we show several other examples in this section.

Example 10.1: Get all the different town names from the PLAYERS table.

SELECT TOWN FROM PLAYERS GROUP BY TOWN

The intermediate result from the GROUP BY clause could look similar to this:

TOWN	PLAYERNO	NAME
Midhurst Inglewood Plymouth	{44, 8}	{Parmenter, Hope,} {Collins} {Baker, Newcastle} {Bailey} {Miller} {Collins, Moorman}

Explanation: All rows with the same TOWN form one group. Each row in the intermediate result has one value in the TOWN column, whereas all other columns can contain multiple values. To indicate that these columns are special, the values are placed between brackets. We show those columns in this way for illustrative purposes only; MySQL probably would solve this internally in a different way. Furthermore, these two columns *cannot* be presented like this because a column that is not grouped is completely omitted from the end result. We return to this topic later in the chapter.

The end result of the statement is:

TOWN Stratford Midhurst Inglewood Plymouth Douglas Eltham

A frequently used term in this particular context is *grouping*. The GROUP BY clause in the previous statement has one grouping, which consists of only one column—the TOWN column. In this chapter, we sometimes represent this as

follows: The result is grouped by [TOWN]. Later in this chapter, we give examples of groupings with multiple columns and GROUP BY clauses consisting of multiple groupings.

The earlier question could be solved more easily by leaving out the GROUP BY clause and adding DISTINCT to the SELECT clause instead (work this out by yourself). Using the GROUP BY clause becomes interesting when we extend the SELECT clause with aggregation functions.

Example 10.2: For each town, find the number of players.

SELECT TOWN, COUNT(*) FROM PLAYERS GROUP BY TOWN

The result is:

TOWN	COUNT(*)
Stratford	7
Midhurst	1
Inglewood	2
Plymouth	1
Douglas	1
Eltham	2

Explanation: In this statement, the result is grouped by [TOWN]. The COUNT(*) function is now executed against each grouped row (for each town) instead of against all rows.

In this result, the data is clearly aggregated. The individual data of players cannot be displayed anymore, and the data is aggregated by TOWN. The aggregation level of this result is TOWN.

Example 10.3: For each team, get the team number, the number of matches that has been played for that team, and the total number of sets won.

SELECT FROM GROUP BY	MATCHES	COUNT(*),	SUM(WON)
The	result is:		
TEAMNO	COUNT(*)	SUM(WON)	
1	8	15	
2	5	9	

Explanation: This statement contains one grouping, consisting of the TEAMNO column.

Example 10.4: For each team that is captained by a player resident in Eltham, get the team number and number of matches that has been played for that team.

```
SELECT
         TEAMNO, COUNT(*)
FROM
         MATCHES
WHERE
         TEAMNO IN
        (SELECT
                   TEAMNO
                   TEAMS INNER JOIN PLAYERS
         FROM
                   ON TEAMS.PLAYERNO = PLAYERS.PLAYERNO
                   TOWN = 'Eltham')
         WHFRF
GROUP BY TEAMNO
   The result is:
TEAMNO COUNT(*)
_____
           _ _ _ _ _ _
                5
     2
```

The column on which the result has been grouped might also appear in the SELECT clause as a parameter within an aggregation function. This does not happen often, but it is allowed.

Example 10.5: Get each different penalty amount, followed by the number of times that the amount occurs in the PENALTIES table, and also show the result of that amount multiplied by the number.

SELECT AMOUNT, COUNT(*), SUM(AMOUNT) FROM PENALTIES GROUP BY AMOUNT

The PENALTIES table is grouped on the AMOUNT column first. The intermediate result could be presented as follows:

PAYMENTNO	PLAYERNO	PAYMENT_DATE	AMOUNT
{5, 6}	{44, 8}	$\{1980-12-08, 1980-12-08\}$	25.00
{7}	{44}	{1982-12-30}	30.00
{4}	{104}	{1984-12-08}	50.00
{2, 8}	{44, 27}	{1981-05-05, 1984-11-12}	75.00
$\{1, 3\}$	{6, 27}	{1980-12-08, 1983-09-10}	100.00

Again, the values of the columns that are not grouped are placed between brackets, and the AMOUNT column shows only one value. However, that is not entirely correct. Behind the scenes, MySQL also creates a group for this column. So the intermediate result should, in fact, be presented as follows:

PAYMENTNO	PLAYERNO	PAYMENT_DATE	AMOUNT
<pre> {5, 6} {7} {4}</pre>	 {44, 8} {44} {104}	{1980-12-08, 1980-12-08} {1982-12-30} {1984-12-08}	{25.00, 25.00} {30.00} {50.00}
{2, 8} {1, 3}	{44, 27} {6, 27}	{1981-05-05, 1984-11-12} {1980-12-08, 1983-09-10}	. , ,

The values in the AMOUNT column are also represented as a group now. Of course, only equal values appear in each group. And because it is a group, aggregation functions can be used.

The result is:

AMOUNT	COUNT(*)	SUM(AMOUNT)
25.00	2	50.00
30.00	1	30.00
50.00	1	50.00
75.00	2	150.00
100.00	2	200.00

However, this book does not present the values of the grouped columns between brackets.

Exercise 10.1: Show the different years in which players joined the club; use the PLAYERS table.

Exercise 10.2: For each year, show the number of players who joined the club.

Exercise 10.3: For each player who has incurred at least one penalty, give the player number, average penalty amount, and number of penalties.

Exercise 10.4: For each team that has played in the first division, give the team number, number of matches, and total number of sets won.

10.3 GROUPING ON TWO OR MORE COLUMNS

A GROUP BY clause can contain two or more columns—or, in other words, a grouping can consist of two or more columns. The next two examples illustrate this topic.

Example 10.6: For the MATCHES table, get all the different combinations of team numbers and player numbers.

SELECT TEAMNO, PLAYERNO FROM MATCHES GROUP BY TEAMNO, PLAYERNO

The result is grouped not on one column, but on two. All rows with the same team number and the same player number form a group.

TEAMNO	PLAYERNO	MATCHNO	WON	LOST
1	2	{6} {1, 2, 3}	{1} {3, 2, 3}	{3} {1, 3, 0}
1	8	{8}	{0}	{3}
1 1	44 57	{4} {7}	{3} {3}	{2} {0}
1 2	83 8	{5} {13}	{0} {0}	{3} {3}
2	27	{9}	{3}	{2}
2 2	104 112	{10} {11, 12}	{3} {2, 1}	{2} {3, 3}

The intermediate result from the GROUP BY clause is:

The end result is:

TEAMNO	PLAYERNO
1	2
1	6
1	8
1	44
1	57
1	83
2	8
2	27
2	104
2	112

The sequence of the columns in the GROUP BY clause has no effect on the end result of a statement. The following statement, therefore, is equivalent to the previous one:

SELECT TEAMNO, PLAYERNO FROM MATCHES GROUP BY PLAYERNO, TEAMNO

As an example, let us add some aggregation functions to the previous SELECT statement:

355

SELECT TEAMNO, PLAYERNO, SUM(WON), COUNT(*), MIN(LOST) FROM MATCHES GROUP BY TEAMNO, PLAYERNO

The result is:

TEAMNO	PLAYERNO	SUM(WON)	COUNT(*)	MIN(LOST)
1	2	1	1	3
1	6	8	3	0
1	8	0	1	3
1	44	3	1	2
1	57	3	1	0
1	83	0	1	3
2	8	0	1	3
2	27	3	1	2
2	104	3	1	2
2	112	3	2	3

In this example, the grouping is equal to [TEAMNO, PLAYERNO], and the aggregation level of the result is the combination of team number and player number. This aggregation level is lower than that of a statement in which the grouping is equal to [TEAMNO] or [TOWN].

Example 10.7: For each player who has ever incurred at least one penalty, get the player number, name, and total amount of penalties incurred.

```
SELECT P.PLAYERNO, NAME, SUM(AMOUNT)
FROM PLAYERS AS P INNER JOIN PENALTIES AS PEN
ON P.PLAYERNO = PEN.PLAYERNO
GROUP BY P.PLAYERNO, NAME
```

The result is:

P.PLAYERNO	NAME	SUM(AMOUNT)
6	Parmenter	100.00
8	Newcastle	25.00
27	Collins	175.00
44	Baker	130.00
104	Moorman	50.00

Explanation: This example also has a grouping consisting of two columns. The statement would have given the same result if the PEN.PLAYERNO column had also been added to the grouping. Work this out by yourself.

Exercise 10.5: For each combination of won-lost sets in the MATCHES table, get the number of matches won.

Exercise 10.6: Group the matches on town of player and division of team, and get the sum of the number of sets won for each combination of town-division.

Exercise 10.7: For each player who lives in Inglewood, get the name, initials, and number of penalties incurred by him or her.

Exercise 10.8: For each team, get the team number, division, and total number of sets won.

10.4 GROUPING ON EXPRESSIONS

Until now, we have shown only examples in which the result was grouped on one or more columns, but what happens when we group on expressions? See the next two examples.

Example 10.8: For each year in the PENALTIES table, get the number of penalties paid.

SELECT YEAR(PAYMENT_DATE), COUNT(*) FROM PENALTIES GROUP BY YEAR(PAYMENT_DATE)

The intermediate result from the GROUP BY clause is:

YEAR(PAYMENT_DATE)	PAYMENTNO	PLAYERNO	PAYMENT_DATE	AMOUNT
1980	{1, 5, 6}	{6, 44, 8}	{1980-12-08,	{100.00,
			1980-12-08,	25,00,
			1980-12-08}	25,00}
1981	{2}	{44}	{1981-05-05}	{75,00}
1982	{7}	{44}	{1982-12-30}	{30,00}
1983	{3}	{27}	{1983-09-10}	$\{100,00\}$
1984	{4, 8}	{104, 27}	{1984-12-08,	{50,00,
			1984-11-12}	75,00}

The result is:

YEAR(PAYMENT_DATE)	COUNT(*)
1980	3
1981	1
1982	1
1983	1
1984	2

Explanation: The result is now grouped on the values of the scalar expression YEAR(PAYMENT_DATE). Rows for which the value of the expression YEAR(PAYMENT_DATE) is equal form a group.

Example 10.9: Group the players on the basis of their player numbers. Group 1 should contain the players with number 1 up to and including 24. Group 2 should contain the players with numbers 25 up to and including 49, and so on. For each group, get the number of players and the highest player number.

```
SELECT TRUNCATE(PLAYERNO/25,0), COUNT(*), MAX(PLAYERNO)
FROM PLAYERS
GROUP BY TRUNCATE(PLAYERNO/25,0)
```

The result is:

TRUNCATE(PLAYERNO/25,0)	COUNT(*)	MAX(PLAYERNO)
0	4	8
1	4	44
2	1	57
3	2	95
4	3	112

The scalar expression on which rows are grouped can be rather complex. This can consist of system variables, user variables, functions, and calculations. Even certain scalar subqueries are allowed.

Exercise 10.9: Group the players on the length of their names and get the number of players for each length.

Exercise 10.10: For each match, determine the difference between the number of sets won and lost, and group the matches on that difference.

Exercise 10.11: For each combination of year-month in the COMMITTEE_ MEMBERS table, get the number of committee members who started in that year and that month.

10.5 GROUPING OF NULL VALUES

If grouping is required on a column that contains null values, all these null values form one group because a GROUP BY clause applies a vertical comparison. This is in accordance with the rules described in Section 9.5.

Example 10.10: Find the different league numbers.

SELECT LEAGUENO FROM PLAYERS GROUP BY LEAGUENO

The result is:

LEAGUENO

Explanation: Players 7, 28, 39, and 95 do not have a league number and, therefore, form one group (the last row) in the end result.

10.6 GROUPING WITH SORTING

In many cases, a select block containing a GROUP BY clause ends with an ORDER BY clause. And many times the columns specified in that ORDER BY clause are the same as the ones specified in the GROUP BY clause. These statements can be simplified by combining the two clauses.

Example 10.11: For each team, get the number of matches and sort the result in descending order by team number.

The obvious formulation is:

SELECT TEAMNO, COUNT(*) FROM MATCHES GROUP BY TEAMNO ORDER BY TEAMNO DESC
 The result is:

 TEAMNO
 COUNT(*)

 2
 5

 1
 8

Explanation: The specification DESC is a *sort direction* and indicates that the result must be sorted in a descending order. This statement can be simplified by including the specification DESC in the GROUP BY clause.

SELECT TEAMNO, COUNT(*) FROM MATCHES GROUP BY TEAMNO DESC

If the result must have an ascending sort direction, ASC (ascending) must be specified.

10.7 GENERAL RULES FOR THE GROUP BY CLAUSE

This section describes a number of important rules for select blocks with a GROUP BY clause.

Rule 1: Section 9.7 gives several rules for the use of aggregation functions in the SELECT clause. For many SQL products, the following rule applies: If a select block has a GROUP BY clause, any column specification in the SELECT clause must exclusively occur as a parameter of an aggregation function, or in the list of columns given in the GROUP BY clause, or in both. Therefore, for most products, the following statement is incorrect because the TOWN column appears in the SELECT clause, yet it is *not* the parameter of an aggregation function and does not occur in the list of columns by which the result is grouped.

```
SELECT TOWN, COUNT(*)
FROM PLAYERS
GROUP BY SEX
```

This restriction is because the result of an aggregation function always consists of one value for each group. The result of a column specification on which grouping is performed also always consists of one value per group. These results are compatible. In contrast, the result of a column specification on which *no* grouping is performed consists of a set of values. This would not be compatible with the results of the other expressions in the SELECT clause. This rule does not apply for MySQL. The previous query will return the following result:

TOWN	COUNT(*)
Stratford	9
Inglewood	5

The value of the second column is understandable—it is the number of players per sex. But the answer of the first column is unexpected. Why does it show Stratford in the first row and Inglewood in the second? This is strange because, for each sex, there can be multiple towns. The answer is that MySQL itself determines the values to be returned. Those values are selected almost randomly. We can enforce this rule if we add the setting ONLY_FULL_GROUP_BY to the SQL_MODE system variable.

Therefore, we strongly recommend that you do not formulate this type of SQL statement, and instead adhere to this rule that applies to most SQL products.

Rule 2: In most examples, the expressions used to form groups also occur in the SELECT clause. However, that is not necessary. An expression that occurs in the GROUP BY clause *can* appear in the SELECT clause.

Rule 3: An expression that is used to form groups can also occur in the SELECT clause within a compound expression. See the next example.

Example 10.12: Get the list with the different penalty amounts in cents.

SELECT CAST(AMOUNT * 100 AS SIGNED INTEGER) AS AMOUNT_IN_CENTS FROM PENALTIES GROUP BY AMOUNT

The result is:

AMOUNT_IN_CENTS 2500 3000 5000 7500 10000

Explanation: A grouping is performed on a simple expression consisting of the column name AMOUNT. In the SELECT clause, that same AMOUNT column occurs within a compound expression. This is allowed.

No matter how complex a compound expression is, if it occurs in a GROUP BY clause, it can be included in its entirety only in the SELECT clause. For example, if the compound expression PLAYERNO * 2 occurs in a GROUP BY clause, the expressions PLAYERNO * 2, (PLAYERNO * 2) - 100 and MOD(PLAYERNO * 2, 3) - 100 can occur in the SELECT clause. On the other hand, the expressions PLAYERNO, 2 * PLAYERNO, PLAYERNO * 100, and 8 * PLAYERNO * 2 are not allowed.

Rule 4: If an expression occurs more than once in a GROUP BY clause, double expressions are simply removed. The GROUP BY clause GROUP BY TOWN, TOWN is converted to GROUP BY TOWN. Also GROUP BY SUBSTR(TOWN,1,1), SEX, SUBSTR(TOWN,1,1) is converted to GROUP BY SUBSTR(TOWN,1,1), SEX.

Rule 5: Section 9.4 described the cases in which the use of DISTINCT in the SELECT clause is superfluous. The rules given in that section apply to SELECT statements without a GROUP BY clause. A different rule exists for SELECT statements with a GROUP BY clause: DISTINCT (if used outside an aggregation function) is superfluous when the SELECT clause includes all the columns specified in the GROUP BY clause. The GROUP BY clause groups the rows in such a way that the column(s) on which they are grouped no longer contain duplicate values.

Exercise 10.12: Describe why the following statements are incorrect:

 SELECT PLAYERNO, DIVISION FROM TEAMS GROUP BY PLAYERNO
 SELECT SUBSTR(TOWN,1,1), NAME FROM PLAYERS GROUP BY TOWN, SUBSTR(NAME,1,1)
 SELECT PLAYERNO * (AMOUNT + 100) FROM PENALTIES GROUP BY AMOUNT + 100

Exercise 10.13: In which of the following statements is DISTINCT superfluous?

1. SELECT DISTINCT PLAYERNO FROM TEAMS GROUP BY PLAYERNO

```
2. SELECT DISTINCT COUNT(*)
FROM MATCHES
GROUP BY TEAMNO
```

3. SELECT DISTINCT COUNT(*) FROM MATCHES WHERE TEAMNO = 2 GROUP BY TEAMNO

10.8 THE GROUP_CONCAT FUNCTION

A special aggregation function that MySQL supports is the GROUP_CONCAT function. The value of this function is equal to all values of the specified column belonging to a group. These values are placed behind each other, separated by commas, and are presented as one long alphanumeric value.

Example 10.13: For each team, get the team number and list of players who played matches for that team.

SELECT TEAMNO, GROUP_CONCAT(PLAYERNO) FROM MATCHES GROUP BY TEAMNO

The result is:

TEAMNO GROUP_CONCAT(PLAYERNO) 1 6,8,57,2,83,44,6,6 2 27,104,112,112,8

The $GROUP_CONCAT$ function can also be used on the column on which the result is grouped.

Example 10.14: For each team, get the team number and for each player, who played matches for that team, get that same team number.

SELECT TEAMNO, GROUP_CONCAT(TEAMNO) FROM MATCHES GROUP BY TEAMNO The result is:

TEAMNO GROUP_CONCAT(TEAMNO) 1 1,1,1,1,1,1,1,1 2 2,2,2,2,2 If a select block contains no GROUP BY clause, the $GROUP_CONCAT$ function is processed on all the values of a column.

Example 10.15: Get all the payment numbers.

SELECT GROUP_CONCAT(PAYMENTNO) FROM PENALTIES

The result is:

GROUP_CONCAT(BETALINGSNR)

1,2,3,4,5,6,7,8

The length of the alphanumeric value of a GROUP_CONCAT function is restricted. The system variable GROUP_CONCAT_MAX_LEN indicates the maximum length. This variable has a standard value of 1,024 and can be adjusted with a SET statement.

Example 10.16: Reduce the length of the GROUP_CONCAT function to seven characters and execute the statement of the previous example.

SET @@GROUP_CONCAT_MAX_LEN=7

SELECT TEAMNO, GROUP_CONCAT(TEAMNO) FROM MATCHES GROUP BY TEAMNO

The result is:

TEAMNO GROUP_CONCAT(TEAMNO) 1 1,1,1,1 2 2,2,2,2

10.9 COMPLEX EXAMPLES WITH GROUP BY

Consider the following examples that illustrate the extensive possibilities of the GROUP BY clause.

Example 10.17: What is the average total amount of penalties for players who live in Stratford and Inglewood?

```
SELECT
         AVG(TOTAL)
FROM
        (SELECT
                   PLAYERNO, SUM(AMOUNT) AS TOTAL
         FROM
                   PENALTIES
         GROUP BY PLAYERNO) AS TOTALS
WHERE
         PLAYERNO IN
        (SELECT
                   PLAYERNO
         FROM
                   PLAYERS
         WHERE
                  TOWN = 'Stratford' OR TOWN = 'Inglewood')
   The result is:
AVG(TOTAL)
_____
        85
```

Explanation: The intermediate result of the subquery in the FROM clause is a table consisting of two columns, called PLAYERNO and TOTAL, and five rows (players 6, 8, 27, 44, and 104). This table is passed on to the WHERE clause, in which a subquery is used to select players from Stratford and Inglewood (players 6, 8, and 44). Finally, the average is calculated in the SELECT clause of the column TOTAL.

Example 10.18: For each player (who incurred penalties and is captain), get the player number, name, number of penalties that he or she incurred, and number of teams that he or she captains.

SELECT	PLAYERS.PL	_AYERNO, NAME, _TEAMS	NUMBER_O	F_PENALTIES,
FROM	PLAYERS,			
	(SELECT		IT(*) AS N	UMBER_OF_PENALTIES
	GROUP BY	PLAYERNO) AS N	UMBER_PEN	ALTIES,
				IUMBER_OF_TEAMS
	FROM	ΓEAMS		
	GROUP BY	PLAYERNO) AS N	IUMBER_TEA	MS
WHERE	PLAYERS.P	AYERNO = NUME	ER_PENALT	IES.PLAYERNO
AND	PLAYERS.P	AYERNO = NUME	ER_TEAMS.	PLAYERNO
The	result is:			
PLAYERNO) NAME	NUMBER_OF_F	PENALTIES	NUMBER_OF_TEAMS
6	6 Parmente	r	1	1
27	′ Collins		2	1

Explanation: The FROM clause contains two subqueries that both have a GROUP BY clause.

We could more easily formulate the previous statement by including subqueries in the SELECT clause, which eliminates the need for GROUP BY clauses. See the next example, and note that the only difference is that all players appear in the result.

SELECT	PLAYERS.	PLAYERNO, NAME,
	(SELECT	COUNT(*)
	FROM	PENALTIES
	WHERE	PLAYERS.PLAYERNO =
		PENALTIES.PLAYERNO) AS NUMBER_OF_PENALTIES,
	(SELECT	COUNT(*)
	FROM	TEAMS
	WHERE	PLAYERS.PLAYERNO =
		TEAMS.PLAYERNO) AS NUMBER_OF_TEAMS
FROM	PLAYERS	

Example 10.19: Get the player number and total number of penalties for each player who played a match.

SELECT DISTINCT M.PLAYERNO, NUMBERP FROM MATCHES AS M LEFT OUTER JOIN (SELECT PLAYERNO, COUNT(*) AS NUMBERP FROM PENALTIES GROUP BY PLAYERNO) AS NP ON M.PLAYERNO = NP.PLAYERNO

Explanation: In this statement, the subquery creates the following intermediate result (this is the NP table):

Next, this table is joined with the MATCHES table. We execute a left outer join, so no players will disappear from this table. The final result is:

PLAYERNO	NUMBERP
2	?
6	1
8	1
27	2
44	3
57	?
83	?
104	1
112	?

Example 10.20: Group the penalties on the basis of payment date. Group 1 should contain all penalties between January 1, 1980, and June 30, 1982; group 2 should contain all penalties between July 1, 1981, and December 31, 1982; and group 3 should contain all penalties between January 1, 1983, and December 31, 1984. For each group, get the sum of all penalties.

```
SELECT
         GROUPS.PGROUP, SUM(P.AMOUNT)
FROM
         PENALTIES AS P,
        (SELECT 1 AS PGROUP, '1980-01-01' AS START,
                '1981-06-30' AS END
         UNION
         SELECT 2, '1981-07-01', '1982-12-31'
         UNION
         SELECT 3, '1983-01-01', '1984-12-31') AS GROUPS
WHERE
         P.PAYMENT_DATE BETWEEN START AND END
GROUP BY GROUPS.PGROUP
ORDER BY GROUPS.PGROUP
   The result is:
GROUP SUM(P.AMOUNT)
_ _ _ _ _
       225.00
    1
    2
               30.00
    3
              225.00
```

Explanation: In the FROM clause, a new (virtual) table is created in which the three groups have been defined. This GROUPS table is joined with the PENAL-TIES table. A BETWEEN operator is used to join the two tables. Penalties with a payment date that falls outside these groups will not be included in the result.

Example 10.21: For each penalty, get the penalty amount plus the sum of that amount and the amounts of all penalties with a lower payment number (cumulative value).

367

SELECT P1.PAYMENTNO, P1.AMOUNT, SUM(P2.AMOUNT) FROM PENALTIES AS P1, PENALTIES AS P2 WHERE P1.PAYMENTNO >= P2. PAYMENTNO GROUP BY P1. PAYMENTNO, P1.AMOUNT ORDER BY P1. PAYMENTNO

For convenience, assume that the PENALTIES table consists of the following three rows only (you can create this, too, by temporarily removing all penalties with a number greater than 3):

PAYMENTNO	PLAYERNO	PAYMENT_DATE	AMOUNT
1	6	1980-12-08	100
2	44	1981-05-05	75
3	27	1983-09-10	100

The desired result is:

PAYMENTNO	AMOUNT	SUM
1	100	100
2	75	175
3	100	275

The intermediate result of the FROM clause (showing only the columns PAYMENTNO and AMOUNT):

P1.PAYNO	P1.AMOUNT	P2.PAYNO	P2.AMOUNT
1	100	1	100
1	100	2	75
1	100	3	100
2	75	1	100
2	75	2	75
2	75	3	100
3	100	1	100
3	100	2	75
3	100	3	100

The intermediate result of the WHERE clause:

P1.PAYNO	P1.AMOUNT	P2.PAYNO	P2.AMOUNT
1	100	1	100
2	75	1	100
2	75	2	75
3	100	1	100
3	100	2	75
3	100	3	100

The intermediate result of the GROUP BY clause:

P1.PAYNO	P1.AMOUNT	P2.PAYNO	P2.AMOUNT
1	100	{1}	{100}
2	75	$\{1, 2\}$	{100, 75}
3	100	$\{1, 2, 3\}$	{100, 75, 100}

The intermediate result of the SELECT clause:

P1.PAYNO	P1.AMOUNT	SUM(P2.AMOUNT)
1	100	100
2	75	175
3	100	275

This final result is equal to the desired table.

Most joins in this book (and in the real world) are equi joins. Non-equi joins are rare. However, the previous statement shows an example where non-equi joins can be useful and the powerful statements they can formulate.

Example 10.22: For each penalty, get the payment number, penalty amount, and percentage that the amount forms of the sum of all amounts (use the same PENALTIES table as in the previous example).

SELECT	•	P1.PAYMENTNO, P1.AMOUNT,
		(P1.AMOUNT * 100) / SUM(P2.AMOUNT)
FROM		PENALTIES AS P1, PENALTIES AS P2
GROUP	ΒY	P1.PAYMENTNO, P1.AMOUNT
ORDER	ΒY	P1.PAYMENTNO

The intermediate result of the FROM clause is equal to that of the previous example. However, the intermediate result of the GROUP BY clause differs:

P1.PAYNO	P1.AMOUNT	P2.PAYNO	P2.AMOUNT
1	100	$\{1, 2, 3\}$	$\{100, 75, 100\}$
2	75	$\{1, 2, 3\}$	$\{100, 75, 100\}$
3	100	$\{1, 2, 3\}$	{100, 75, 100}

The intermediate result of the SELECT clause is:

P1.PAYNO	P1.AMOUNT	(P1.AMOUNT * 100) / SUM(P2.AMOUNT)
1	100	36.36
2	75	27.27
3	100	36.36

Determine yourself whether this is the final result.

Exercise 10.14: How many players live in a town, on average?

Exercise 10.15: For each team, get the team number, division, and number of players that played matches for that team.

Exercise 10.16: For each player, get the player number, name, sum of all penalties that he or she incurred, and number of teams from the first division that he or she captains.

Exercise 10.17: For each team captained by a player who lives in Stratford, get the team number and number of players who have won at least one match for that team.

Exercise 10.18: For each player, get the player number, name, and difference between the year in which he or she joined the club and the average year of joining the club.

Exercise 10.19: For each player, get the player number, name, and difference between the year in which he or she joined the club and the average year in which players who live in the same town joined the club.

10.10 GROUPING WITH WITH ROLLUP

The GROUP BY clause has many features to group data and calculate aggregated data, such as the total number of penalties or the sum of all penalties. However, all statements return results in which all data is on the same level of aggregation. But what if we want to see data belonging to different aggregation levels within one statement? Imagine that with one statement we want to see the total penalty amount for each player, followed by the total penalty amount for all players. The forms of the GROUP BY clauses discussed so far do not make this possible. To achieve the desired result, more than two groupings within one GROUP BY clause are required. By adding the specification WITH ROLLUP to the GROUP BY clause, it becomes possible.

Example 10.23: For each player, find the sum of all his or her penalties, plus the sum of all penalties.

Use the UNION operator as a way to combine these two groupings into one statement.

SELECT	PLAYERNO, SUM(AMOUNT)	
FROM	PENALTIES	
GROUP BY	PLAYERNO	
UNION		
SELECT	NULL, SUM(AMOUNT)	
FROM	PENALTIES	

The result is:

PLAYERNO	SUM(AMOUNT)	
6	100.00	
8	25.00	
27	175.00	
44	130.00	
104	50.00	
?	480.00	

Explanation: The rows of this intermediate result in which the PLAYERNO column is filled form the result of the first select block. The rows in which PLAYERNO is equal to null make up the result of the second select block. The first five rows contain data on the aggregation level of the player numbers, and the last row contains data on the aggregation level of all rows.

The specification WITH ROLLUP has been introduced to simplify this kind of statement. WITH ROLLUP can be used to ask for multiple groupings with one GROUP BY clause. Using this approach, the previous statement would then be

SELECTPLAYERNO, SUM(AMOUNT)FROMPENALTIESGROUP BYPLAYERNO WITH ROLLUP

Explanation: The result of this statement is the same as the previous one. The specification WITH ROLLUP indicates that after the result has been grouped on [PLAYERNO], another grouping is needed—in this case, on all rows.

To further define this concept, imagine that the expressions E_1 , E_2 , E_3 , and E_4 are specified in a GROUP BY clause. The grouping performed is $[E_1, E_2, E_3, E_4]$. When we add the specification WITH ROLLUP to this GROUP BY clause, an entire set of groupings is performed: $[E_1, E_2, E_3, E_4]$, $[E_1, E_2, E_3]$, $[E_1, E_2]$, $[E_1]$, and finally []. The specification [] means that all rows are grouped into one group. The specified grouping is seen as the highest aggregation level that is asked and also indicates that all higher aggregation levels must be calculated again. To aggregate upward is called rollup. So the result of this statement contains data on five different levels of aggregation. If an expression occurs in the SELECT clause in which the result of a certain grouping is not grouped, the null value is placed in the result.

Example 10.24: For each combination of sex-town, get the number of players, total number of players per sex, and total number of players in the entire table.

SELECT SEX, TOWN, COUNT(*) FROM PLAYERS GROUP BY SEX, TOWN WITH ROLLUP

The result is:

SEX	TOWN	COUNT(*)
М	Stratford	7
М	Inglewood	1
М	Douglas	1
М	?	9
F	Midhurst	1
F	Inglewood	1
F	Plymouth	1
F	Eltham	2
F	?	5
?	?	14

Explanation: This result has three levels of aggregation. Rows 1, 2, 3, 5, 6, 7, and 8 form the lowest level and have been added because of the grouping [SEX, TOWN]; rows 4 and 9 have been added because of the grouping [SEX]; and the last row forms the highest level of aggregation and has been added because of the grouping []. It contains the total number of players.

Exercise 10.20: For each team, get the number of matches played and also the total number of matches.

Exercise 10.21: Group the matches by the name of the player and division of the team, and execute a ROLLUP. Then for each group, get the name of the player, division of the team, and total number of sets won.

10.11 Answers

- 10.1 SELECT JOINED FROM PLAYERS GROUP BY JOINED
- 10.2 SELECT JOINED, COUNT(*) FROM PLAYERS GROUP BY JOINED
- 10.3 SELECT PLAYERNO, AVG(AMOUNT), COUNT(*) FROM PENALTIES GROUP BY PLAYERNO
- 10.4 SELECT TEAMNO, COUNT(*), SUM(WON) FROM MATCHES WHERE TEAMNO IN (SELECT TEAMNO FROM TEAMS WHERE DIVISION = 'first') GROUP BY TEAMNO
- 10.5 SELECT WON, LOST, COUNT(*) FROM MATCHES WHERE WON > LOST GROUP BY WON, LOST ORDER BY WON, LOST
- 10.6 SELECT P.TOWN, T.DIVISION, SUM(WON) FROM (MATCHES AS M INNER JOIN PLAYERS AS P ON M.PLAYERNO = P.PLAYERNO) INNER JOIN TEAMS AS T ON M.TEAMNO = T.TEAMNO GROUP BY P.TOWN, T.DIVISION ORDER BY P.TOWN
- 10.7 SELECT NAME, INITIALS, COUNT(*) FROM PLAYERS AS P INNER JOIN PENALTIES AS PEN ON P.PLAYERNO = PEN.PLAYERNO WHERE P.TOWN = 'Inglewood' GROUP BY P.PLAYERNO, NAME, INITIALS
- **10.8** SELECT T.TEAMNO, DIVISION, SUM(WON) FROM TEAMS AS T, MATCHES AS M WHERE T.TEAMNO = M.TEAMNO GROUP BY T.TEAMNO, DIVISION

373

- 10.9 SELECT LENGTH(RTRIM(NAME)), COUNT(*) FROM PLAYERS GROUP BY LENGTH(RTRIM(NAME))
- 10.10 SELECT ABS(WON LOST), COUNT(*) FROM MATCHES GROUP BY ABS(WON - LOST)
- 10.11 SELECT YEAR(BEGIN_DATE), MONTH(BEGIN_DATE), COUNT(*) FROM COMMITTEE_MEMBERS GROUP BY YEAR(BEGIN_DATE), MONTH(BEGIN_DATE) ORDER BY YEAR(BEGIN_DATE), MONTH(BEGIN_DATE)
- **10.12 1.** Although this column appears in the SELECT clause, the result of the DIVISION column has not been grouped.
 - **2.** The NAME column cannot appear like this in the SELECT clause because the result has not been grouped on the full NAME column.
 - **3.** The PLAYERNO column appears in the SELECT clause, although the result has not been grouped; furthermore, the column does not appear as a parameter of an aggregation function.
- 10.13 1. Superfluous
 - 2. Not superfluous
 - 3. Superfluous
- 10.14 SELECT AVG(NUMBERS) FROM (SELECT COUNT(*) AS NUMBERS FROM PLAYERS GROUP BY TOWN) AS TOWNS
- 10.15 SELECT FROM FROM TEAMS.TEAMNO, DIVISION, NUMBER_PLAYERS TEAMS LEFT OUTER JOIN (SELECT TEAMNO, COUNT(*) AS NUMBER_PLAYERS FROM MATCHES GROUP BY TEAMNO) AS M ON (TEAMS.TEAMNO = M.TEAMNO)

10.16 SELECT PLAYERS.PLAYERNO, NAME, SUM_AMOUNT, NUMBER_TEAMS FROM (PLAYERS LEFT OUTER JOIN (SELECT PLAYERNO, SUM(AMOUNT) AS SUM_AMOUNT FROM PENALTIES GROUP BY PLAYERNO) AS TOTALS ON (PLAYERS.PLAYERNO = TOTALS.PLAYERNO)) LEFT OUTER JOIN PLAYERNO, COUNT(*) AS NUMBER_TEAMS (SELECT FROM TEAMS WHERE DIVISION = 'first' GROUP BY PLAYERNO) AS NUMBERS ON (PLAYERS.PLAYERNO = NUMBERS.PLAYERNO)

10.17 SELECT TEAMNO, COUNT(DISTINCT PLAYERNO) FROM MATCHES WHERE TEAMNO IN (SELECT TEAMNO FROM PLAYERS AS P INNER JOIN TEAMS AS T ON P.PLAYERNO = T.PLAYERNO AND TOWN = 'Stratford') AND WON > LOST

GROUP BY TEAMNO

- 10.18 SELECT PLAYERNO, NAME, JOINED AVERAGE FROM PLAYERS, (SELECT AVG(JOINED) AS AVERAGE FROM PLAYERS) AS T
- 10.19 SELECT PLAYERNO, NAME, JOINED AVERAGE FROM PLAYERS, (SELECT TOWN, AVG(JOINED) AS AVERAGE FROM PLAYERS GROUP BY TOWN) AS TOWNS WHERE PLAYERS.TOWN = TOWNS.TOWN
- 10.20 SELECT TEAMNO, COUNT(*) FROM MATCHES GROUP BY TEAMNO WITH ROLLUP

10.21 SELECT P.NAME, T.DIVISION, SUM(WON) FROM (MATCHES AS M INNER JOIN PLAYERS AS P ON M.PLAYERNO = P.PLAYERNO) INNER JOIN TEAMS AS T ON M.TEAMNO = T.TEAMNO GROUP BY P.NAME, T.DIVISION WITH ROLLUP