BSCCS2001: Practice/Graded Assignment with Solutions
Week 2

Modules covered:

1. Attribute Types, Relation Schema and Instance, Keys, Relational Query Languages
2. Operations, Select, Project, Union, Difference, Intersection, Cartesian Product
3. Natural Join, Aggregate Operations
4. Introduction to SQL - History of SQL, Data Definition Language (DDL), Basic Query Structure (DML)
5. Additional Basic Operations, Set Operations, Null Values, Aggregate Functions

1. Consider the three relations given below.
   Note that the primary keys are underlined. [MCQ: 1 point]
   employees *(employee_num, employee_name, contact_num, salary)*
   taskAssignment *(employee_num, task_num, task_duration)*
   tasks *(task_num, location)*
   Select the list of possible foreign key(s) for the given relations.

   ○ *employee_num*

   ✓ *employee_num, task_num*

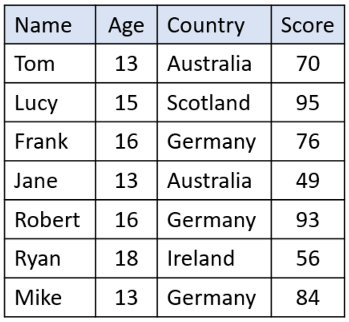
   ○ *task_num*

   ○ *task_num, task_duration*

Solution: The possible foreign keys are as follows:

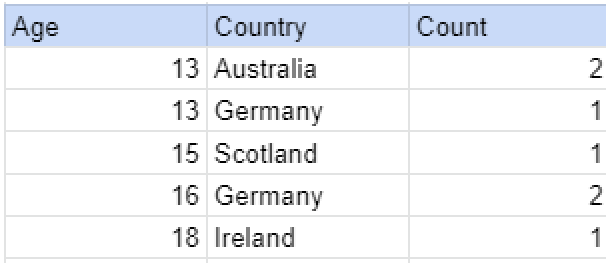
- *employee_num* of **taskAssignment** is a foreign key that refers to the relation **employees**.
- *task_num* of **taskAssignment** is a foreign key that refers to the relation **tasks**.

2. Using the table **Students**, choose the correct SQL statement that will return the resultant table given in Figure 2. [MCQ: 2 points]



Name	Age	Country	Score
Tom	13	Australia	70
Lucy	15	Scotland	95
Frank	16	Germany	76
Jane	13	Australia	49
Robert	16	Germany	93
Ryan	18	Ireland	56
Mike	13	Germany	84

Figure 1: Table Students



Age	Country	Count
13	Australia	2
13	Germany	1
15	Scotland	1
16	Germany	2
18	Ireland	1

Figure 2: Resultant table

○ `SELECT Age, Country, COUNT(*) FROM Students GROUP BY Name;`

○ `SELECT Age, Country, COUNT(*) FROM Students GROUP BY Age, Score;`

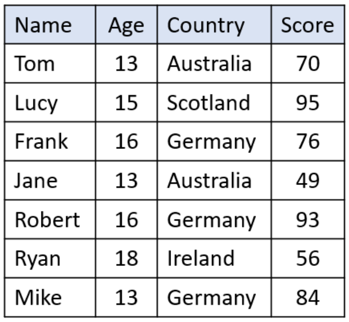
√ `SELECT Age, Country, COUNT(*) FROM Students GROUP BY Age, Country;`

○ `SELECT Age, Country, COUNT(*) FROM Students GROUP BY Score, Country;`

Solution: The GROUP BY clause is used to group data based on specific values of the given attribute. Here, the tuples are grouped based on attributes Age and Country and and tuples with same values like {13, Australia} are grouped into one tuple. Similarly, tuples with same values like {16, Germany} have also been grouped.

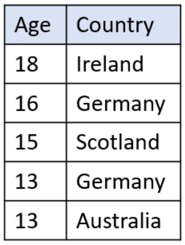
3. Using the table **Students**, choose the correct SQL statement that will return the resultant table given in Figure 4.

[MCQ: 2 points]



Name	Age	Country	Score
Tom	13	Australia	70
Lucy	15	Scotland	95
Frank	16	Germany	76
Jane	13	Australia	49
Robert	16	Germany	93
Ryan	18	Ireland	56
Mike	13	Germany	84

Figure 3: Table Students



Age	Country
18	Ireland
16	Germany
15	Scotland
13	Germany
13	Australia

Figure 4: Resultant table

○ `SELECT Age, Country FROM Students ORDER BY Score ASC;`

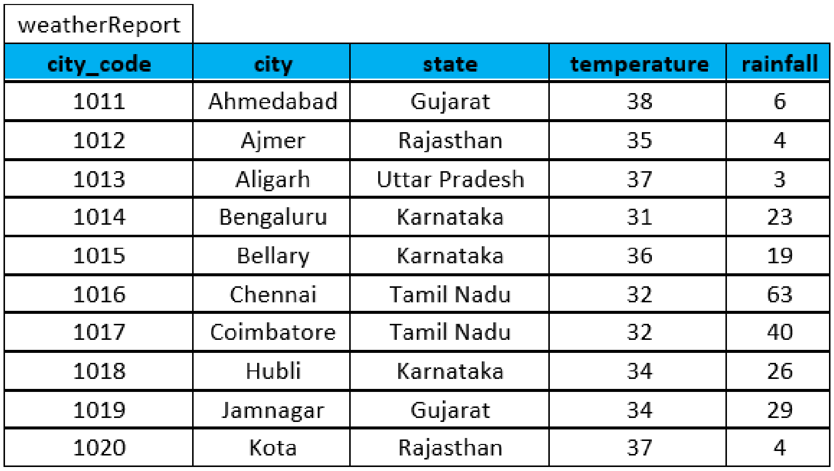
○ `SELECT DISTINCT Age, Country FROM Students ORDER BY Age ASC;`

○ `SELECT DISTINCT Age, Country FROM Students ORDER BY Score DESC;`

√ `SELECT DISTINCT Age, Country FROM Students ORDER BY Age DESC;`

Solution: DISTINCT keyword is used to eliminate duplicate records based on the specified attribute(s).
ORDER BY clause is used to sort the data in ascending or descending order, based on one or more columns.
Here, the resultant table will be fetched by retrieving distinct Age and Country, based on sorting the scores in descending order.

Using the table in Figure 5 to answer the questions 4 and 5.



weatherReport				
city_code	**city**	**state**	**temperature**	**rainfall**
1011	Ahmedabad	Gujarat	38	6
1012	Ajmer	Rajasthan	35	4
1013	Aligarh	Uttar Pradesh	37	3
1014	Bengaluru	Karnataka	31	23
1015	Bellary	Karnataka	36	19
1016	Chennai	Tamil Nadu	32	63
1017	Coimbatore	Tamil Nadu	32	40
1018	Hubli	Karnataka	34	26
1019	Jamnagar	Gujarat	34	29
1020	Kota	Rajasthan	37	4

Figure 5: Table **weatherReport**

4. Based on the data given in the table in Figure 5, identify the appropriate query to find the city having minimum rainfall. [MCQ: 3 points]

○
```
SELECT city
FROM weatherReport
HAVING rainfall = MAX(rainfall);
```

○
```
SELECT city
FROM weatherReport
WHERE rainfall = MAX(rainfall);
```

○
```
SELECT t1.city
FROM weatherReport AS t1, weatherReport AS t2
WHERE t1.rainfall < t2.rainfall;
```

√
```
SELECT DISTINCT city
FROM weatherReport
EXCEPT
SELECT DISTINCT t1.city
FROM weatherReport AS t1, weatherReport AS t2
WHERE t1.rainfall > t2.rainfall;
```

Solution: The `HAVING` keyword must be used along with `GROUP BY` keyword. Thus, SQL statement in option 1 is wrong.
The aggregate function like `MAX` must be used in condition with `HAVING` keyword. Thus, SQL statement in option 2 is wrong.

The SQL statement finds out all cities that have rainfall lesser than that of some of the cities. Thus, SQL statement in option 3 is wrong.

The statement:

```
SELECT DISTINCT city FROM weatherReport
```

selects all the cities.

The statement:

```
SELECT DISTINCT t1.city FROM weatherReport AS t1,
weatherReport AS t2 WHERE t1.rainfall > t2.rainfall;
```

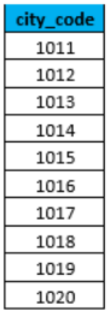
selects all the cities which have rainfall higher than some of the cities. In other words, it extracts all rows except the row with minimum rainfall. The `EXCEPT` keyword returns the rows which are there in the first set of rows, but not there in the second set of rows, i.e. the row that has minimum rainfall. Finally, the SQL statement projects the *city*. Thus, option 4 is correct.

Note: `EXCEPT` is available in the PostgreSQL and SQLite database while `MINUS` is available in MySQL and Oracle.

5. Based on the data given in the table in Figure 5, identify the output for the following SQL statement. [MCQ: 2 points]

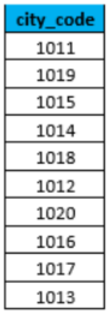
```
SELECT city_code
FROM weatherReport
ORDER BY state, city;
```

○ Output:



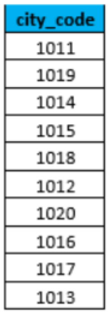
city_code
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020

✓ Output:



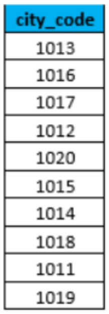
city_code
1011
1019
1015
1014
1018
1012
1020
1016
1017
1013

○ Output:



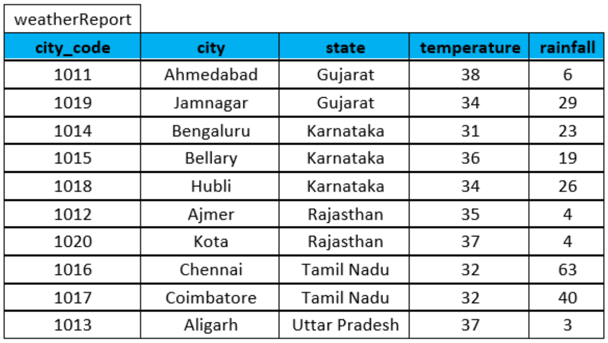
city_code
1011
1019
1014
1015
1018
1012
1020
1016
1017
1013

○ Output:



city_code
1013
1016
1017
1012
1020
1015
1014
1018
1011
1019

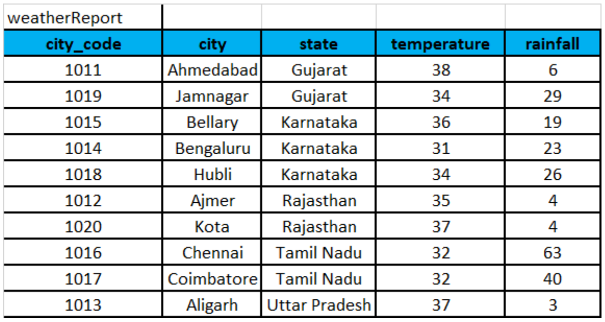
Solution: The SQL statement first sorts the table by the *state*, and the output is as follows:



weatherReport

city_code	city	state	temperature	rainfall
1011	Ahmedabad	Gujarat	38	6
1019	Jamnagar	Gujarat	34	29
1014	Bengaluru	Karnataka	31	23
1015	Bellary	Karnataka	36	19
1018	Hubli	Karnataka	34	26
1012	Ajmer	Rajasthan	35	4
1020	Kota	Rajasthan	37	4
1016	Chennai	Tamil Nadu	32	63
1017	Coimbatore	Tamil Nadu	32	40
1013	Aligarh	Uttar Pradesh	37	3

Next, for each *state*, sort the table by *city* and the output is as follows:



weatherReport

city_code	city	state	temperature	rainfall
1011	Ahmedabad	Gujarat	38	6
1019	Jamnagar	Gujarat	34	29
1015	Bellary	Karnataka	36	19
1014	Bengaluru	Karnataka	31	23
1018	Hubli	Karnataka	34	26
1012	Ajmer	Rajasthan	35	4
1020	Kota	Rajasthan	37	4
1016	Chennai	Tamil Nadu	32	63
1017	Coimbatore	Tamil Nadu	32	40
1013	Aligarh	Uttar Pradesh	37	3

Finally, project the column *city_code*, which is option 2.

6. Which of the following statement(s) are **TRUE**? [MSQ: 1 point]

○ All superkeys are candidate keys

√ All candidate keys are superkeys

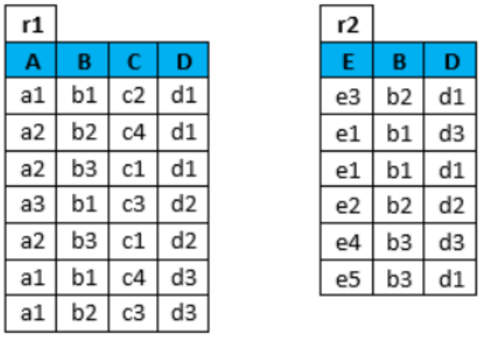
√ A foreign key can be a primary key

○ All superkeys are primary keys

Solution:

- A superkey K is a candidate key if K is minimal. Thus, all candidate keys must be superkeys, but all superkeys need not be candidate keys.
- One of the candidate keys is selected to be the primary key. Thus, the primary key is a candidate key and obviously a superkey. However, minimal superkeys are candidate keys, and one of the candidate keys becomes primary key.
- It is possible that a foreign key can be a primary key.

7. Consider two relations as shown in Figure 6: [MSQ: 3 points]

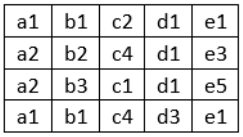


r1			
A	B	C	D
a1	b1	c2	d1
a2	b2	c4	d1
a2	b3	c1	d1
a3	b1	c3	d2
a2	b3	c1	d2
a1	b1	c4	d3
a1	b2	c3	d3

r2		
E	B	D
e3	b2	d1
e1	b1	d3
e1	b1	d1
e2	b2	d2
e4	b3	d3
e5	b3	d1

Figure 6: Relations **r1** and **r2**

Identify the correct operation(s) that result(s) in the output shown in Figure 7.



a1	b1	c2	d1	e1
a2	b2	c4	d1	e3
a2	b3	c1	d1	e5
a1	b1	c4	d3	e1

Figure 7: Output tuples

○ $r1 \times r2$

√ $r1 \bowtie r2$

○ $\sigma_{r1.B=r2.B \ \wedge \ r1.D=r2.D}(r1 \times r2)$

√ $\pi_{A,r1.B,C,r1.D,E}(\sigma_{r1.B=r2.B \ \wedge \ r1.D=r2.D}(r1 \times r2))$

Solution: Since relations **r1** and **r2** have B and D as common attributes and the given output relation is the set of tuples that have corresponding pairs of B and D values equal in **r1** and **r2**, it follows that the given output is a natural join of **r1** and **r2**. Hence, answer $r1 \bowtie r2$ is correct.
The answer $\pi_{A,r1.B,C,r1.D,E}(\sigma_{r1.B=r2.B \ \wedge \ r1.D=r2.D}(r1 \times r2))$ is also a correct answer, as it is equivalent to $r1 \bowtie r2$.

8. Consider a table **department** that has *salary* as an attribute. What will be the output of the following query?

[MSQ: 1 point]

- SELECT *salary* FROM department WHERE *salary* LIKE '30%5_%_';

  ✓ salary with value 305500

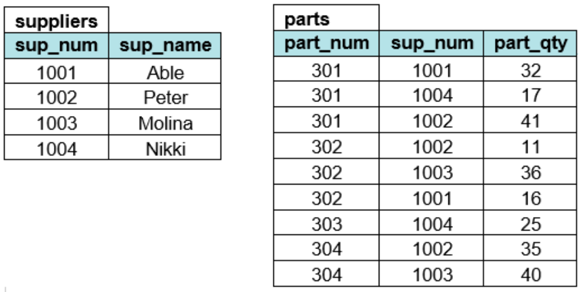
  ✓ salary with value 305005

  ○ salary with value 3050

  ○ salary with value 30050

Solution: The percentage sign (%) represents zero, one, or multiple characters and the underscore sign (_) represents a single character.

Use the tables in Figure 8 to answer the questions 9 and 10.



suppliers

sup_num	sup_name
1001	Able
1002	Peter
1003	Molina
1004	Nikki

parts

part_num	sup_num	part_qty
301	1001	32
301	1004	17
301	1002	41
302	1002	11
302	1003	36
302	1001	16
303	1004	25
304	1002	35
304	1003	40

Figure 8: Table **suppliers** and table **parts**

9. Identify the SQL statement(s) that find(s) the names of suppliers who supply parts with *part_num* 301 but do not supply parts with *part_num* 304. [MSQ: 2 points]

○
```
SELECT sup_name
FROM suppliers s, parts p
WHERE s.sup_num = p.sup_num AND
(part_num = 301 AND part_num <> 304);
```

○
```
SELECT sup_name
FROM suppliers s, parts p
WHERE s.sup_num = p.sup_num and
(part_num = 301 OR part_num <> 304);
```

√
```
SELECT sup_name
FROM suppliers s, parts p
WHERE s.sup_num = p.sup_num AND part_num = 301
EXCEPT
SELECT sup_name
FROM suppliers s, parts p
WHERE s.sup_num = p.sup_num AND part_num = 304;
```

○
```
SELECT sup_name
FROM suppliers s, parts p
WHERE s.sup_num = p.sup_num AND part_num = 301
INTERSECT
SELECT sup_name
FROM suppliers s, parts p
WHERE s.sup_num = p.sup_num AND part_num = 304;
```

Solution: From the problem statement: "find the names of suppliers who supply parts with *part_num* 301 but do not supply parts with *part_num* 304" clearly indicates that the desired operation is 'set difference'. In SQL statement, set difference operation is presented by `EXCEPT` keyword. Thus, option 3 is correct.

10. Let $\{sup\_num\}$ be the primary key of the table **suppliers** and $\{part\_num, sup\_num\}$ be the primary key of the table **parts**. [NAT: 3 points]
Consider the SQL query given below:

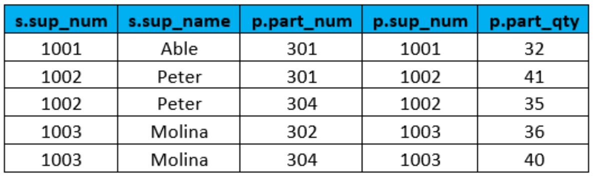
```
SELECT s.sup_num, sum(p.part_qty)
FROM suppliers s, parts p
WHERE p.part_qty > 30 AND s.sup_num = p.sup_num
GROUP BY s.sup_num
```

How many rows will be returned by the above SQL query?
Answer: 3

Solution: As per the given SQL statement, it first performs a Cartesian product between **suppliers** and **parts**, which output all possible combinations from both the tables.
The part of the statement:

```
WHERE p.part_qty > 30 AND s.sup_num = p.sup_num
```

eliminates the rows which do not satisfy the condition. The output is as shown below:

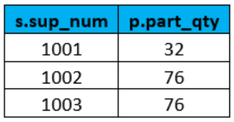


s.sup_num	s.sup_name	p.part_num	p.sup_num	p.part_qty
1001	Able	301	1001	32
1002	Peter	301	1002	41
1002	Peter	304	1002	35
1003	Molina	302	1003	36
1003	Molina	304	1003	40

Finally, the part of the statement:

```
SELECT s.sup_num, sum(p.part_qty) ...GROUP BY s.sup_num
```

results in:



s.sup_num	p.part_qty
1001	32
1002	76
1003	76

Thus, the result has 3 rows.