Fundamentals of Database Technologies

Mock exam 1 solutions

Part A

Question 1

The relational database paradigm is a model for the storage and retrieval of data. It specifies that data should be stored in tables (relations) which have rows (records) and columns (attributes). The benefits for business are that data is stored in a unified format which is interoperable with other stakeholders' data storage systems. The benefits for data scientists are that data from different stakeholders is expressed by the same model, allowing analysis skills to be shared.

Full marks require: understanding what a paradigm is; tables and relations. Any good selection of benefits is appropriate.

Question 2

First normal form (1NF) specifies that each attribute within a record should be atomic; in other words, that each cell (intersection of a row and column) must be a single, indivisible value rather than a list of values.

A relation is in second normal form if a) it is in first normal form, and b) no non-prime attribute is functionally dependent on any proper subset of any candidate key of the relation; in other words, if every non-prime attribute of the relation is functionally dependent on the whole of every candidate key.

Must specify 1NF details too

Definition of FD not necessary

Question 3

- No transitive dependencies between non-key attributes

Question 4

a) A one-to-one relationship can link a maximum of one entity at each end of the relationship, whereas a many-to-one relationship may have more than one entity at one end of the relationship.

b) The jobs end of the relationship has a || symbol, indicating that one entity must be present. The employees end of the relationship has a O < symbol, indicating that zero or many entities may be present. This is therefore a one-to-many relationship.

c) A many-to-many relationship requires a extra table called a junction table, which has one row for each link in the many-to-many relationship.

d) No. In particular, there is no many-to-many relationship between employees and locations because each employee can only have one department and thus one location.

Part B

Question 1

```
SELECT * FROM employees
```

Question 2

```
SELECT first_name, last_name, job_title
FROM employees INNER JOIN jobs
ON employees.job_id = jobs.job_id
```

Question 3

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id
```

Question 4

```
SELECT locations.country_id, COUNT(employee_id)
FROM
employees INNER JOIN departments
ON employees.department_id = departments.department_id
INNER JOIN locations
ON departments.location_id = locations.location_id
GROUP BY locations.country_id
```

Question 5

```
SELECT first_name, last_name, AVG(salary) OVER (PARTITION BY
department_id)
FROM employees
```

Question 6

a)
i)

Both departments will be put into the same group, since grouping is done on department name. This will lead to erroneous results as the departments are actually separate.

ii)

```
SELECT departments.department_name, COUNT(employee_id)
FROM employees INNER JOIN departments
ON employees.department_id = departments.department_id
GROUP BY departments.department_id
```

We are now doing grouping based on the department id, which means that the departments with the same name will be kept separate. We are still able to ask for departments.department_name in the list of columns, since this is functionally dependent on departments.department_id.

b)

i)

In both of these questions it is which TABLE the columns belong to that matters (if you GROUP BY a column in a table you can also return other columns from that table) - not whether the exact column titles are present in the GROUP BY (which is not necessary).

This query will not run, since we are grouping by departments.department_id, but we are asking for employees.department_id in the list of columns. The SQL processor does not know that a row with a particular employees.department_id will always have the same departments.department_id (even though we asked for the join to be done on this row, so we know this to be true).

An equivalent query in the dvdrental database is

```
SELECT payment.staff_id, COUNT(payment_id)
FROM payment INNER JOIN staff
ON payment.staff_id = staff.staff_id
GROUP BY staff.staff_id
```

ii) This query will run, because the columns requested in the columns list (departments.department_id and departments.department_name) are, respectively, identical to and functionally dependent on the column which is being grouped on (departments.department_id). The SQL processor therefore knows that they will have the same value for every row in each group.

An equivalent query in the dvdrental database is

```
SELECT staff.staff_id, staff.first_name, staff.last_name, COUNT(payment_id)
FROM payment INNER JOIN staff
ON payment.staff_id = staff.staff_id
GROUP BY staff.staff_id
```