Fundamentals of Database Technologies
Practice exam 2 solutions

PART A

1

Normal forms prevent database corruption. They protect our data from wasted storage space, update errors (one value not updated) and delete errors (when values we wish to preserve are deleted in order to delete other values). 1NF makes access easier and safer by eliminating delimiters and user parsing. 2NF acts to prevent duplicated data by preventing partial dependencies from the key.

2

Second normal form enforces no partial dependencies from the key, whereas third normal form enforces no transitive dependencies from the key (no dependencies between non-key attributes).

3

One-to-one relationships connect one entity to zero or one other entities. They are encoded by having one entity hold the other's key as an attribute.

One-to-many relationships connect one entity to zero or more other entities. Here the entities on the "many" end must hold the key, since the entity on the "one" end only has room for one without copying rows.

Many-to-many relationships connect each entity to zero or many other entities, bidirectionally. Since neither end could hold a list of keys without duplicating rows, we require an extra "join table" which holds both keys for each link in the relationship.

4

a) One-to-one or one-to-many, because each discount holds a single store id.

b) Many-to-many, because we can see the title-author join table.

PART B

1

```
SELECT * FROM titles;
```

2

```
SELECT title_id, title, price*ytd_sales
FROM titles;
```

3

```
SELECT au_lname, au_fname, AVG(price)
FROM
authors INNER JOIN titleauthor
ON authors.au_id = titleauthor.au_id
INNER JOIN titles
ON titleauthor.title_id = titles. title_id
GROUP BY authors.au_id;
```

4

```
SELECT stores.store_id, COUNT(*)
FROM
discounts LEFT OUTER JOIN stores
ON discounts.store_id = stores.store_id
GROUP BY stores.store_id;
```

Question indicates show all stores, so use LEFT OUTER JOIN.

5

Error in the question: there is no salary column in the employees table. We will use job_lvl instead.

```
SELECT fname, lname, employee.pub_id,
AVG(job_lvl) OVER (PARTITION BY employees.pub_id) AS mean_level
FROM
employees;
```

6

i)

```
SELECT *,
SUM(price*qty) OVER (ORDER BY ord_date) AS cumulative_price
FROM
sales INNER JOIN titles
ON sales.title_id = titles.title_id;
```

ii)

```
SELECT *,
SUM(price*qty) OVER (PARTITION BY titles.title_id ORDER BY ord_date) AS
cumulative_price
```

```
FROM
sales INNER JOIN titles
ON sales.title_id = titles.title_id;
```