

Item: 1 (Ref:Cert-1Z0-071.6.2.4)

In which situation would you use a natural join?

- ☐ The tables being joined do not have primary and foreign keys defined.
- ☐ The tables being joined have matching columns with different names but compatible data types.
- ☐ The tables being joined each have two columns with the same name and incompatible data types, and you want to join on both of the columns.
- ☐ The tables being joined each have two columns with the same name and compatible data types, and you want to join on one of the columns.
- ☐ The tables being joined each have two columns with the same name and compatible data types, and you want to join on both of the columns.

Answer:

The tables being joined each have two columns with the same name and compatible data types, and you want to join on both of the columns.

Explanation:

Natural joins are used to join tables based on all columns in the tables that have the same name. When implementing a natural join, columns with the same names must have compatible data types. Natural joins would be appropriate if the tables being joined have two columns with the same name and compatible data types, and you want to join on both of the columns.

Natural joins can be implemented regardless of whether the tables being joined have primary and foreign keys defined. Because this has no effect on natural joins, the option stating that primary and foreign key definitions are relevant is incorrect.

The option stating that a natural join would be used to join tables having matching columns with different names but compatible data types is incorrect. The syntax of the `NATURAL JOIN` statement does not include column names (only table names), and the join is created using all columns with the same names. Columns with different names cannot be joined using a natural join.

Natural joins cannot be used to join columns with incompatible data types. All columns with the same name must have compatible data types, or an error occurs. Therefore, the option stating that a natural join would be used to join columns with incompatible data types is incorrect.

The option stating that a natural join would be used when the tables being joined have two columns with the same name and compatible data types, and you want to join on one of the columns is incorrect. A natural join will use all common columns in the join condition. If you need to restrict the join condition to only one of the same-named columns, you would not use a natural join. Rather, you would use a simple join with a `USING` clause, a simple join with an `ON` clause, or specify the join condition in the `WHERE` clause with Oracle proprietary syntax.

Item: 2 (Ref:Cert-1Z0-071.6.2.1)

Click the Exhibit(s) button to examine the structures of the `PATIENT`, `PHYSICIAN`, and `ADMISSION` tables.

You want to create a report containing the patient name, physician name, and admission date for all admissions.

Which two `SELECT` statements could you use? (Choose two. Each correct answer is a separate solution.)

- ☐ SELECT x.last_name || ', ' || x.first_name as "Patient Name",
y.last_name || ', ' || y.first_name as "Physician Name",
z.admit_date
FROM patient x, physician y, admission z
WHERE x.patient_id = z.patient_id
AND y.physician_id = z.physician_id;
- ☐ SELECT x.last_name || ', ' || x.first_name as "Patient Name",
y.last_name || ', ' || y.first_name as "Physician Name",
z.admit_date
FROM patient x JOIN physician y
ON (x.patient_id = z.patient_id)
JOIN admission z
ON (y.physician_id = z.physician_id);
- ☐ SELECT x.last_name || ', ' || x.first_name as "Patient Name",
y.last_name || ', ' || y.first_name as "Physician Name",
z.admit_date
FROM patient x JOIN admission z
ON (x.patient_id = z.patient_id)
JOIN physician y
ON (y.physician_id = z.physician_id);
- ☐ SELECT last_name || ', ' || first_name as "Patient Name",
last_name || ', ' || first_name as "Physician Name",
admit_date
FROM patient NATURAL JOIN admission NATURAL JOIN physician;

Answer:

```
SELECT x.last_name || ', ' || x.first_name as "Patient Name",
y.last_name || ', ' || y.first_name as "Physician Name",
z.admit_date
FROM patient x, physician y, admission z
WHERE x.patient_id = z.patient_id
AND y.physician_id = z.physician_id;

SELECT x.last_name || ', ' || x.first_name as "Patient Name",
y.last_name || ', ' || y.first_name as "Physician Name",
z.admit_date
FROM patient x JOIN admission z
ON (x.patient_id = z.patient_id)
JOIN physician y
ON (y.physician_id = z.physician_id);
```

PATIENT

PATIENT_ID	NUMBER	NOT NULL, Primary Key
LAST_NAME	VARCHAR2 (30)	NOT NULL
FIRST_NAME	VARCHAR2 (25)	NOT NULL
DOB	DATE	
INS_CODE	NUMBER	

PHYSICIAN

PHYSICIAN_ID	NUMBER	NOT NULL, Primary Key
LAST_NAME	VARCHAR2 (30)	NOT NULL
FIRST_NAME	VARCHAR2 (25)	NOT NULL
LICENSE_NO	NUMBER(7)	NOT NULL
HIRE_DATE	DATE	

ADMISSION

ADMISSION_ID	NUMBER	NOT NULL, Primary Key
PATIENT_ID	NUMBER	NOT NULL, References PATIENT_ID column of the PATIENT table
PHYSICIAN_ID	NUMBER	NOT NULL, References PHYSICIAN_ID column of the PHYSICIAN table
ADMIT_DATE	DATE	
DISCHG_DATE	DATE	
ROOM_ID	NUMBER	Foreign key to ROOM_ID of the ROOM table

Explanation:

To create a report containing the patient name, physician name, and admission date, you must join all three tables using equijoins. When joining two or more tables using equijoins, you can use standard Oracle syntax by including the join condition in the WHERE clause, or you can use SQL: 1999 syntax using a JOIN . . . ON, a JOIN . . . USING, or a NATURAL JOIN. In the given scenario, you could either use the statement that joins the three tables by specifying the join condition in the WHERE clause, or you could use the statement that joins the three tables using the JOIN . . . ON syntax.

When joining more than two tables using an ON clause, the joins are evaluated from left to right. Additionally, a column cannot be referenced until after the column's table has been specified. Therefore, the correct statement using the JOIN . . . ON syntax must join the PATIENT and ADMISSION tables first, and then join the PHYSICIAN table.

The option that uses the JOIN . . . ON syntax but joins the PATIENT and PHYSICIAN tables first is incorrect because the ON clauses do not correspond with the appropriate join.

The option that joins the tables using natural joins is incorrect. Natural joins join tables based on all columns in the two tables that have the same name. Because the PATIENT and PHYSICIAN tables both contain columns named FIRST_NAME and last_name, a natural join would join the two tables based on both of these columns and would only return patients and physicians who had the same name.

The following statement would create the same result with a USING clause:

```
SELECT x.last_name || ', ' || x.first_name as "Patient Name",
       y.last_name || ', ' || y.first_name as "Physician Name",
       z.admit_date
FROM patient x JOIN admission z
  USING(patient_id)
JOIN physician y
```

USING (physician_id);

Item: 3 (Ref:Cert-1Z0-071.6.4.3)

Click the **Exhibit(s)** button to examine the structures of the EMPLOYEE and TASK tables.

You need to produce a report containing all employees and all tasks. An employee must be included on the report even if he has no tasks assigned. All tasks, whether assigned to an employee or not, must also be included on the report.

Which SELECT statement should you use?

- ☐ SELECT e.emp_lname, e.emp_fname, t.task_description, t.est_compl_date
FROM employee e, task t
WHERE e.employee_id = t.employee_id;
- ☐ SELECT e.emp_lname, e.emp_fname, t.task_description, t.est_compl_date
FROM employee e, task t
WHERE e.employee_id (+) = t.employee_id;
- ☐ SELECT e.emp_lname, e.emp_fname, t.task_description, t.est_compl_date
FROM employee e, task t
WHERE e.employee_id = t.employee_id (+);
- ☐ SELECT e.emp_lname, e.emp_fname, t.task_description, t.est_compl_date
FROM employee e, task t
WHERE e.employee_id (+) = t.employee_id (+);
- ☐ None of the options will produce the desired result.

Answer:

None of the options will produce the desired result.

EMPLOYEE

EMPLOYEE_ID	NUMBER	NOT NULL, Primary Key
EMP_LNAME	VARCHAR2(25)	
EMP_FNAME	VARCHAR2(25)	
DEPT_ID	NUMBER	Foreign key to DEPT_ID column of DEPARTMEN
JOB_ID	NUMBER	Foreign key to JOB_ID column of JOB table
MGR_ID	NUMBER	References EMPLOYEE_ID column
SALARY	NUMBER(9,2)	
HIRE_DATE	DATE	

TASK

PROJECT_ID	NUMBER	NOT NULL, Primary Key, Foreign key to PROJ column of PRODUCT table
TASK_ID	NUMBER	NOT NULL, Primary Key
TASK_DESCRIPTION	VARCHAR2(100)	
EST_COMPL_DATE	DATE	
EMPLOYEE_ID	NUMBER	Foreign key to EMPLOYEE_ID column of EMPLO table

Explanation:

For the given scenario, none of the options will produce the desired result. Because you needed to include all rows from both tables, a full outer join must be used, and none of the given options correctly implements a full outer join.

Outer joins may be created in one of two ways. You can either create a full outer join using the SQL: 1999 syntax, or you can use the `FULL OUTER JOIN` syntax as in this `SELECT` statement:

```
SELECT e.emp_lname, e.emp_fname, t.task_description, t.est_compl_date
FROM employee e FULL OUTER JOIN task t
ON (e.employee_id = t.employee_id);
```

Using Oracle proprietary syntax, you cannot include the outer join operator (+) on both sides of the join condition. To implement an outer join you must use two `SELECT` statements, one performing a left outer join and the other performing a right outer join, and combine the results of these `SELECT` statements using the `UNION` operator, as shown in the following example:

```
SELECT e.emp_lname, e.emp_fname, t.task_description, t.est_compl_date
FROM employee e, task t
WHERE e.employee_id (+) = t.employee_id
UNION
SELECT e.emp_lname, e.emp_fname, t.task_description, t.est_compl_date
FROM employee e, task t
WHERE e.employee_id = t.employee_id (+);
```

The option including `WHERE e.employee_id = t.employee_id` as the join condition is incorrect because this implements an equijoin, or inner join, and will only return matching rows from the two tables.

Both of the options that use the outer join operator on one side of the join condition are incorrect. Neither would return unmatched rows from both tables.

The option that includes the outer join operator on both sides of the join condition is also incorrect because it is not valid to use the outer join operator on both sides of a join condition.

Item: 4 (Ref:Cert-1Z0-071.6.4.4)

Examine the data from the `po_header` and `po_detail` tables.

PO_HEADER

PO_NUM	PO_DATE	SUPPLIER_ID	PO_TERMS	PO_TOTAL
10052	03-JUL-2001	2	NET30	2030.00
10053	03-JUL-2001	2	NET30	54.55
10054	03-JUL-2001	1	NET60	3805.00
10055	03-JUL-2001	1	NET60	125.00
10056	03-JUL-2001	1	NET60	85.72

PO_DETAIL

PO_NUM	PO_LINE_ID	PRODUCT_ID	QUANTITY	UNIT_PRICE
10052	1	1	100	10.30
10052	2	2	100	10.00
10054	1	1	50	72.10
10054	2	1	10	10.00
10054	3	3	10	10.00
10057	1	1	75	54.30

You need to produce a report to identify any po_header rows that have no matching po_detail rows and any po_detail rows that have no matching po_header record.

Which SELECT statement should you execute?

- ☐ SELECT h.po_num, d.po_num, d.po_line_id
FROM po_header h FULL OUTER JOIN po_detail d
ON (h.po_num = d.po_num)
WHERE h.po_num IS NULL
OR d.po_line_id IS NULL;
- ☐ SELECT h.po_num, d.po_num, d.po_line_id
FROM po_header h LEFT OUTER JOIN po_detail d
ON (h.po_num = d.po_num)
WHERE d.po_num IS NULL;
- ☐ SELECT h.po_num, d.po_num, d.po_line_id
FROM po_header h FULL OUTER JOIN po_detail d
ON (h.po_num = d.po_num)
WHERE h.po_num IS NULL
AND d.po_line_id IS NULL;
- ☐ SELECT h.po_num, d.po_num, d.po_line_id
FROM po_header h RIGHT OUTER JOIN po_detail d
ON (h.po_num = d.po_num)
WHERE h.po_num IS NULL;

Answer:

```
SELECT h.po_num, d.po_num, d.po_line_id
FROM po_header h FULL OUTER JOIN po_detail d
ON (h.po_num = d.po_num)
WHERE h.po_num IS NULL
OR d.po_line_id IS NULL;
```

Explanation:

You should use the following query to produce the desired report:

```
SELECT h.po_num, d.po_num, d.po_line_id
FROM po_header h FULL OUTER JOIN po_detail d
ON (h.po_num = d.po_num)
WHERE h.po_num IS NULL
OR d.po_line_id IS NULL;
```

In this report, you want to join the two tables, but retrieve only the unmatched rows from both tables. A full outer join will retrieve all matching rows and all unmatched rows from both tables. To eliminate the matching rows, you can exclude `po_header` rows that have matching lines using the condition `d.po_line_id IS NULL` and the `po_detail` rows that have matching headers using the condition `h.po_num IS NULL`. These two conditions would need to be joined using an `OR` logical operator to ensure you retrieve the desired rows.

The `SELECT` statement that implements a left outer join is incorrect because it only returns `po_header` rows that do not have matching `po_detail` rows.

The `SELECT` statement that implements a full outer join but uses the `AND` logical operator in the `WHERE` clause condition is incorrect because it will return no rows. The rows desired will meet one of these conditions, but not both of them.

The `SELECT` statement that implements a right outer join is incorrect because it only returns `po_detail` rows that do not have matching `po_header` rows.

Item: 5 (Ref:Cert-1Z0-071.6.2.2)

Examine the structures of the `product` and `supplier` tables:

`product`

```
-----
PRODUCT_ID NUMBER
PRODUCT_NAME VARCHAR2(25)
SUPPLIER_ID NUMBER
CATEGORY_ID NUMBER
QTY_PER_UNIT NUMBER
UNIT_PRICE NUMBER(7,2)
QTY_IN_STOCK NUMBER
QTY_ON_ORDER NUMBER
REORDER_LEVEL NUMBER
```

`supplier`

```
-----
SUPPLIER_ID NUMBER
SUPPLIER_NAME VARCHAR2(25)
ADDRESS VARCHAR2(30)
CITY VARCHAR2(25)
REGION VARCHAR2(10)
POSTAL_CODE VARCHAR2(11)
```

You want to create a query that will return an alphabetical list of products including the name of each product's supplier. Only products in the `product` table that have a supplier assigned should be included in your report.

Which two queries could you use? (Choose two. Each correct answer is a separate solution.)

☐ `SELECT p.product_name, s.supplier_name`

```
FROM product p
LEFT OUTER JOIN supplier s
ON p.supplier_id = s.supplier_id
ORDER BY p.product_name;
```

☐

```
SELECT p.product_name, s.supplier_name
FROM product p
JOIN supplier s
ON (supplier_id)
ORDER BY p.product_name;
```

☐

```
SELECT product_name, supplier_name
FROM product
NATURAL JOIN supplier
ORDER BY product_name;
```

☐

```
SELECT p.product_name, s.supplier_name
FROM product p
JOIN supplier s
USING (p.supplier_id)
ORDER BY p.product_name;
```

☐

```
SELECT product_name, supplier_name
FROM product
JOIN supplier
USING (supplier_id)
ORDER BY product_name;
```

Answer:

```
SELECT product_name, supplier_name
FROM product
NATURAL JOIN supplier
ORDER BY product_name;

SELECT product_name, supplier_name
FROM product
JOIN supplier
USING (supplier_id)
ORDER BY product_name;
```

Explanation:

To produce the needed list of products, you should join the `product` and `supplier` tables using an equijoin. An equijoin joins two tables by a column that contains a matching value. Several methods exist for performing equijoins. In this situation, you could use one of two statements:

- a statement that implements a natural join
- a statement that implements a simple join containing a `USING` clause with no table alias

Natural joins join two tables by all columns with the same name. Because the `supplier_id` column is the only column with the same name in both tables, a natural join will perform an equijoin based on this column. The `USING` clause creates an equijoin by specifying a column name (or column names) common to both tables and, in this scenario, will perform an equijoin of the two tables using the `supplier_id` columns from each table.

Outer joins join two tables on a matching column, but include unmatched rows from one or both of the joined tables. The statement implementing a left outer join would include all rows from the `product` table and matching rows from the `supplier` table. Because you only wanted to include products that were assigned a supplier, the statement that implements the left outer join is incorrect.

The option that implements a simple join with an `ON` clause is incorrect. The `ON` clause can be used to produce an equijoin, but this statement contains incorrect syntax for the `ON` clause. When using the `ON` clause to produce an equijoin, the join condition should be specified with a traditional join predicate, not a single column reference.

A correct implementation of an equijoin containing the `ON` clause would be:

```
SELECT p.product_name, s.supplier_name
FROM product p JOIN supplier s
ON (p.supplier_id = s.supplier_id);
```

<fo

Item: 6 (Ref:Cert-1Z0-071.6.4.1)

Examine the data from the `class` and `instructor` tables.

CLASS

CLASS_ID	CLASS_NAME	HOURS_CREDIT	INSTRUCTOR_ID
1	Introduction to Accounting	3	4
2	Computer Basics	3	1
3	Tax Accounting Principles	3	4
4	American History	3	2
5	Basic Engineering	3	

INSTRUCTOR

INSTRUCTOR_ID	LAST_NAME	FIRST_NAME
1	Chao	Ling
2	Vanderbilt	Herbert
3	Wigley	Martha
4	Page	Albert

You have been asked to produce a report of all instructors, including the classes taught by each instructor. All instructors must be included on the report, even if they are not currently assigned to teach classes.

Which two `SELECT` statements could you use? (Choose two. Each correct answer is a separate solution.)

- ☐ `SELECT i.last_name, i.first_name, c.class_name`
`FROM instructor i, class c;`
- ☐ `SELECT i.last_name, i.first_name, c.class_name`
`FROM class c LEFT OUTER JOIN instructor i`
`ON (i.instructor_id = c.instructor_id)`
`ORDER BY i.instructor_id;`
- ☐ `SELECT i.last_name, i.first_name, c.class_name`
`FROM instructor i, class c`
`WHERE i.instructor_id = c.instructor_id (+)`
`ORDER BY i.instructor_id;`
- ☐ `SELECT i.last_name, i.first_name, c.class_name`
`FROM instructor i LEFT OUTER JOIN class c`
`ON (i.instructor_id = c.instructor_id)`

```
ORDER BY i.instructor_id;
```

- ☐ `SELECT i.last_name, i.first_name, c.class_name
FROM instructor i, class c
WHERE i.instructor_id (+) = c.instructor_id
ORDER BY i.instructor_id;`
- ☐ `SELECT i.last_name, i.first_name, c.class_name
FROM instructor i NATURAL JOIN class c
ON (i.instructor_id = c.instructor_id);`

Answer:

```
SELECT i.last_name, i.first_name, c.class_name  
FROM instructor i, class c  
WHERE i.instructor_id = c.instructor_id (+)  
ORDER BY i.instructor_id;  
  
SELECT i.last_name, i.first_name, c.class_name  
FROM instructor i LEFT OUTER JOIN class c  
ON (i.instructor_id = c.instructor_id)  
ORDER BY i.instructor_id;
```

Explanation:

To produce the desired report, you must use an outer join condition to include all instructors from the `instructor` table, even if they have no corresponding classes in the `class` table. To create an outer join, either Oracle proprietary syntax or SQL: 1999 syntax can be used. To produce the needed report, you could use either of these approaches:

- the `SELECT` statement that implements an outer join in the `WHERE` clause with `WHERE i.instructor_id = c.instructor_id (+)`
- the `SELECT` statement that implements a left outer join with `FROM instructor i LEFT OUTER JOIN class c`

The statement that implements a simple join but does not include either a `WHERE` clause or an `ON` clause is incorrect. Because no join condition is specified, all rows in the `instructor` table will be joined with all rows in the `class` table. This creates a Cartesian product and is not what you desired.

The statement that uses a `LEFT OUTER JOIN` but lists the `class` table first is incorrect. Because the `class` table is listed to the left of the join, all rows in the `class` table are retrieved, even if there is no match in the `instructor` table. This is exactly opposite of what you needed.

The option that uses Oracle proprietary syntax with `WHERE i.instructor_id (+) = c.instructor_id` as the join condition is also incorrect because the outer join operator is on the wrong side of the join condition.

The option that implements a natural join and includes an `ON` clause is incorrect. This statement will generate an error because neither an `ON` nor a `USING` clause can be used with the `NATURAL JOIN` syntax.

Item: 7 (Ref:Cert-1Z0-071.6.2.5)

Click the Exhibit(s) button to examine the structures of the `donor`, `donation`, and `donor_level` tables.

You want to produce a report of all donors, including each donor's giving level. The donor level should be

determined based on the amount pledged by the donor.

Which SELECT statement will join these three tables and implements a non-equijoin?

- ☐ SELECT d.donor_name, dl.level_description
FROM donor d, donor_level dl
WHERE amount_pledged BETWEEN dl.min_donation AND dl.max_donation;
- ☐ SELECT d.donor_name, dl.level_description
FROM donor d JOIN donation dn
USING (donor_id) JOIN donor_level dl
ON (dn.amount_pledged BETWEEN dl.min_donation AND dl.max_donation);
- ☐ SELECT d.donor_name, dl.level_description
FROM donor d, donation dn, donor_level dl
WHERE dn.amount_pledged BETWEEN dl.min_donation AND dl.max_donation;
- ☐ SELECT d.donor_name, dl.level_description
FROM donor d JOIN donation dn JOIN donor_level dl
ON (donor_id) AND
ON (dn.amount_pledged BETWEEN dl.min_donation AND dl.max_donation);
- ☐ This join cannot be accomplished because the donor_level and donation tables have no common column.

Answer:

```
SELECT d.donor_name, dl.level_description
FROM donor d JOIN donation dn
USING (donor_id) JOIN donor_level dl
ON (dn.amount_pledged BETWEEN dl.min_donation AND dl.max_donation);
```

DONOR

DONOR_ID	NUMBER	NOT NULL, Primary Key
DONOR_NAME	VARCHAR2 (50)	
ADDRESS	VARCHAR2 (30)	
CITY	VARCHAR2 (25)	
REGION	VARCHAR2 (10)	
POSTAL_CODE	VARCHAR2 (11)	

DONATION

PLEDGE_ID	NUMBER	NOT NULL, Primary Key
DONOR_ID	NUMBER	Foreign key to DONOR_ID column of DONOR t
PEDGE_DT	DATE	
AMOUNT_PLEDGED	NUMBER (7, 2)	
AMOUNT_PAID	NUMBER (7, 2)	
PAYMENT_DT	DATE	

DONOR_LEVEL

LEVEL_ID	NUMBER	NOT NULL, Primary Key
MIN_DONATION	NUMBER (7, 2)	
MAX_DONATION	NUMBER (7, 2)	
LEVEL_DESCRIPTION	VARCHAR2 (30)	

Explanation:

The following `SELECT` statement will join the three tables and implements a non-equijoin:

```
SELECT d.donor_name, dl.level_description
FROM donor d JOIN donation dn
USING (donor_id) JOIN donor_level dl
ON (dn.amount_pledged BETWEEN dl.min_donation AND dl.max_donation);
```

A non-equijoin is represented by the use of an operator other than an equality operator (=). A non-equijoin is used when no corresponding columns exist between the tables in the query, but rather a relationship exists between two columns having compatible data types. Several conditions can be used to define a non-equijoin, including <, <=, >, >=, BETWEEN, and IN. In the given scenario, a non-equijoin relationship exists between the `amount_pledged` column of the donation table and the `min_donation` and `max_donation` columns of the donor_level table. In addition, an equijoin relationship exists between the `donor_id` column of the donation table and the `donor_id` column of the donor table. To produce the report of all donors with their corresponding giving levels, you should use the `SELECT` statement that contains an `ON` clause and a `USING` clause. First, this statement joins the donor and donation tables using the commonly named column `donor_id`. Then, this result is joined with the donor levels based on a non-equijoin condition in the `ON` clause, namely `ON (dn.amount_pledged BETWEEN dl.min_donation AND dl.max_donation)`.

The `SELECT` statement that only includes the donor and donor_level tables in the `FROM` clause is incorrect. The donation table must be included in the relationship because it contains the `amount_pledged` column.

The `SELECT` statement that uses traditional Oracle syntax to implement the join in the `WHERE` clause but only includes one `WHERE` clause condition, `WHERE dn.amount_pledged BETWEEN dl.min_donation AND dl.max_donation`, is also incorrect. This join condition provides for no relationship between the donation and donor tables, and it would be impossible to associate a donor name with each donation.

The `SELECT` statement that specifies two `ON` clauses is incorrect because it has invalid syntax. When the `JOIN...ON` syntax is used, the `ON` clause must follow its corresponding `JOIN`.

Item: 8 (Ref:Cert-1Z0-071.6.2.6)

Evaluate this SQL statement:

```
SELECT c.customer_id, o.order_id, o.order_date, p.product_name
FROM customer c, curr_order o, product p
WHERE customer.customer_id = curr_order.customer_id
AND o.product_id = p.product_id
ORDER BY o.order_amount;
```

This statement fails when executed.

Which change will correct the problem?

- ☐ Use the table name in the `ORDER BY` clause.
- ☐ Remove the table aliases from the `WHERE` clause.
- ☐ Include the `order_amount` column in the `SELECT` list.
- ☐ Use the table aliases instead of the table names in the `WHERE` clause.

- ☐ Remove the table alias from the `ORDER BY` clause and use only the column name.

Answer:

Use the table aliases instead of the table names in the `WHERE` clause.

Explanation:

To correct the problem with this `SELECT` statement, you should use the table aliases instead of the table names in the `WHERE` clause. Table aliases are specified for all tables in the `FROM` clause of this `SELECT` statement. After they have been defined, these table aliases must be used. The first join predicate in the `WHERE` clause uses the full table name to qualify each column, and this will result in an error.

The `ORDER BY` clause uses the table alias correctly. Therefore, the option stating you should use the table name in the `ORDER BY` clause is incorrect.

You should not remove the table aliases from the `WHERE` clause, but rather use aliases throughout.

The `order_amount` column does not need to be included in the `SELECT` list. A column not included in the `SELECT` list can be used for ordering.

If no columns had identical names in both tables, you could remove the table alias from the `ORDER BY` clause and use only the column name. However, this would not correct the error in this SQL statement.

While special rules exist for using table aliases, both with Oracle proprietary and SQL: 1999 syntax, using table aliases where possible is recommended. Using table aliases not only makes SQL statements easier to read, but also provides additional performance enhancements.

Item: 9 (Ref: Cert-1Z0-071.6.3.2)

Examine the structures of the `PLAYER` and `TEAM` tables:

```
PLAYER
-----
PLAYER_ID NUMBER(9) PK
LAST_NAME  VARCHAR2(25)
FIRST_NAME VARCHAR2(25)
TEAM_ID    NUMBER
MANAGER_ID NUMBER(9)
```

```
TEAM
-----
TEAM_ID NUMBER PK
TEAM_NAME VARCHAR2(30)
```

For this example, team managers are also players, and the `MANAGER_ID` column references the `PLAYER_ID` column. For players who are managers, `MANAGER_ID` is `NULL`.

Which `SELECT` statement will provide a list of all players, including the player's name, the team name, and the player's manager's name?

- ☐ `SELECT p.last_name, p.first_name, p.manager_id, t.team_name`

```
FROM player p NATURAL JOIN team t;
```

☐ `SELECT p.last_name, p.first_name, p.manager_id, t.team_name`

```
FROM player p JOIN team t
USING (team_id);
```

☐ `SELECT p.last_name, p.first_name, m.last_name, m.first_name, t.team_name`

```
FROM player p
LEFT OUTER JOIN player m ON (p.manager_id = m.player_id)
LEFT OUTER JOIN team t ON (p.team_id = t.team_id);
```

☐ `SELECT p.last_name, p.first_name, m.last_name, m.first_name, t.team_name`

```
FROM player p JOIN player m
ON (p.manager_id = m.player_id)
RIGHT OUTER JOIN team t ON (p.team_id = t.team_id);
```

☐ `SELECT p.last_name, p.first_name, m.last_name, m.first_name, t.team_name`

```
FROM player p
LEFT OUTER JOIN player m ON (p.player_id = m.player_id)
LEFT OUTER JOIN team t ON (p.team_id = t.team_id);
```

Answer:

```
SELECT p.last_name, p.first_name, m.last_name, m.first_name, t.team_name

FROM player p
LEFT OUTER JOIN player m ON (p.manager_id = m.player_id)
LEFT OUTER JOIN team t ON (p.team_id = t.team_id);
```

Explanation:

The following `SELECT` statement will provide the needed list:

```
SELECT p.last_name, p.first_name, m.last_name, m.first_name, t.team_name
FROM player p
LEFT OUTER JOIN player m ON (p.manager_id = m.player_id)
LEFT OUTER JOIN team t ON (p.team_id = t.team_id);
```

This statement joins the table to itself using `FROM player p LEFT OUTER JOIN player m ON (p.manager_id = m.player_id)` and joins this result to the team table using `LEFT OUTER JOIN team t ON (p.team_id = t.team_id)`.

Both of the `SELECT` statements that return only the `manager_id` and not the manager's name are incorrect because you wanted the list to include the manager's name.

The `SELECT` statement that implements a right outer join is incorrect. This statement will first join the `player` table to itself using an equijoin. This will only include players that have a manager assigned. The team managers would not be included in the list. In addition, it then performs a `RIGHT OUTER JOIN` with the `team` table. This will include any teams that do not have players in the list.

The `SELECT` statement that includes two left outer joins but uses `p.player_id = m.player_id` as the self join condition is incorrect because this joins the `player` table to itself using only the `player_id` column. To create the needed self join condition, you must create the relationship from the `player_id` column to the `manager_id` column.

Item: 10 (Ref:Cert-1Z0-071.6.3.1)

Examine the structures of the patient, physician, and admission tables.

PATIENT

PATIENT_ID	NUMBER	NOT NULL, Primary Key
LAST_NAME	VARCHAR2 (30)	NOT NULL
FIRST_NAME	VARCHAR2 (25)	NOT NULL
DOB	DATE	
INS_CODE	NUMBER	

PHYSICIAN

PHYSICIAN_ID	NUMBER	NOT NULL, Primary Key
LAST_NAME	VARCHAR2 (30)	NOT NULL
FIRST_NAME	VARCHAR2 (25)	NOT NULL
LICENSE_NO	NUMBER (7)	NOT NULL
HIRE_DATE	DATE	

ADMISSION

ADMISSION_ID	NUMBER	NOT NULL, Primary Key
PATIENT_ID	NUMBER	NOT NULL, References PATIENT_ID column of PATIENT table
PHYSICIAN_ID	NUMBER	NOT NULL, References PHYSICIAN_ID column of PHYSICIAN table
ADMIT_DATE	DATE	
DISCHG_DATE	DATE	
ROOM_ID	NUMBER	Foreign key to ROOM_ID column of ROOM table

Which SQL statement will produce a list of all patients who have more than one physician?

- ☐ SELECT p.patient_id
FROM patient p
WHERE p.patient_id IN (SELECT patient_id
FROM admission
GROUP BY patient_id
HAVING COUNT(*) > 1);
- ☐ SELECT DISTINCT a.patient_id
FROM admission a, admission a2
WHERE a.patient_id = a2.patient_id
AND a.physician_id <> a2.physician_id;
- ☐ SELECT patient_id
FROM admission
WHERE COUNT(physician_id) > 1;
- ☐ SELECT patient_id
FROM patient FULL OUTER JOIN physician;

Answer:

```
SELECT DISTINCT a.patient_id
FROM admission a, admission a2
WHERE a.patient_id = a2.patient_id
AND a.physician_id <> a2.physician_id;
```

Explanation:

The following SQL statement will produce the list of all patients who have more than one physician:

```
SELECT DISTINCT a.patient_id
FROM admission a, admission a2
WHERE a.patient_id = a2.patient_id
AND a.physician_id <> a2.physician_id;
```

The equal (=) condition in the `WHERE` clause matches each patient with itself, and the not equal (<>) condition restricts the results to only those rows where the physicians are different. This results in a list of patients with more than one physician. However, duplicates are included. Using the `DISTINCT` keyword eliminates these duplicates.

The statement that uses a subquery in the `WHERE` clause is incorrect. In this statement, the inner query executes first and returns all patients who have been admitted more than once. The outer query then returns all patients who have been admitted more than once. A patient might have been admitted more than one time, but with the same physician. Therefore, this query does not accurately return all patients with more than one physician.

The statement that includes the `COUNT` function in the `WHERE` clause is incorrect and will generate an error because aggregate functions cannot be used in a `WHERE` clause.

The statement that implements a full outer join is incorrect. Full outer joins join tables based on a common value, but include null values from both of the joined tables. In this scenario, the `patient` and `physician` tables have no common column, and no `ON` clause was specified for the join. Therefore, an error will result.

Item: 11 (Ref:Cert-1Z0-071.6.4.2)

Click the Exhibit(s) button to examine the structures of the `EMPLOYEE`, `PROJECT`, and `TASK` tables.

You want to create a report of all employees, including employee name and project name, who are assigned to project tasks. You want to include all projects even if they currently have no tasks defined, and you want to include all tasks, even those not assigned to an employee.

Which joins should you use?

- ☐ a self join on the `EMPLOYEE` table and a left outer join between the `TASK` and `PROJECT` tables
- ☐ a natural join between the `TASK` and `EMPLOYEE` tables and a natural join between the `TASK` and `PROJECT` tables
- ☐ a full outer join between the `TASK` and `EMPLOYEE` tables and a natural join between the `TASK` and `PROJECT` tables
- ☐ a natural join between the `TASK` and `EMPLOYEE` tables and a left outer join between the `TASK` and `PROJECT` tables
- ☐ a full outer join between the `TASK` and `EMPLOYEE` tables and a full outer join between the `TASK` and `PROJECT` tables

- ☐ a left outer join between the TASK and EMPLOYEE tables and a right outer join between the TASK and PROJECT tables

Answer:

a left outer join between the TASK and EMPLOYEE tables and a right outer join between the TASK and PROJECT tables

EMPLOYEE

EMPLOYEE_ID	NUMBER	NOT NULL, Primary Key
EMP_LNAME	VARCHAR2(25)	
EMP_FNAME	VARCHAR2(25)	
DEPT_ID	NUMBER	Foreign key to DEPT_ID column of DEPARTME
JOB_ID	NUMBER	Foreign key to JOB_ID column of JOB table
MGR_ID	NUMBER	References EMPLOYEE_ID column
SALARY	NUMBER(9,2)	
HIRE_DATE	DATE	
DOB	DATE	

PROJECT

PROJECT_ID	NUMBER	NOT NULL, Primary Key
PROJECT_NAME	VARCHAR2(30)	
MGR_ID	NUMBER	Foreign Key to EMPLOYEE_ID column of EMPL
PROJECT_TYPE	VARCHAR2(10)	
BEGIN_DT	DATE	
END_DT	DATE	

TASK

PROJECT_ID	NUMBER	NOT NULL, Primary Key, Foreign key to PRO column of PRODUCT table
TASK_ID	NUMBER	NOT NULL, Primary Key
TASK_DESCRIPTION	VARCHAR2(100)	
EST_COMPL_DATE	DATE	
EMPLOYEE_ID	NUMBER	Foreign key to EMPLOYEE_ID column of EMPL

Explanation:

To produce the desired results, you should use a left outer join between the TASK and EMPLOYEE tables and a right outer join between the TASK and PROJECT tables. An example of this operation would be:

```
SELECT p.project_name, t.task_id, e.employee_id
FROM task t LEFT OUTER JOIN employee e
ON (t.employee_id = e.employee_id)
RIGHT OUTER JOIN project p
```

```
ON (t.project_id = p.project_id);
```

The first join in the `SELECT` statement will be evaluated first. The `TASK` and `EMPLOYEE` tables will be joined, with all rows from the task table being included, even if they have no employees assigned. Then, the next join will be evaluated. This joins the result of the first join to the `PROJECT` table with all projects being included regardless of whether the project has associated tasks. The result will be the desired list.

The option that states you would use a self join is incorrect because you do not need to relate the `EMPLOYEE` table to itself.

All of the options stating you would use a natural join are incorrect. Natural joins perform equijoins, which will not include unmatched rows.

The option stating that you would use two full outer joins is incorrect. If a full outer join were used to join the `TASK` and `EMPLOYEE` tables, all employees would be included, even if they were not assigned to any tasks. You only wanted to return employees that were assigned project tasks.

Item: 12 (Ref:Cert-1Z0-071.6.1.2)

Your database contains the `Customers` and `Orders` tables as follows:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country	Sex
1	Aaron Anderons	Jeff Lebowski	2507 Henry St	Santa Monica	90401	USA	M
2	Amy Lawrence	Walter Sobchak	1199 Paddocks Way	Santa Monica	90401	USA	F
3	Anderson East	Jackie Treehorn	11 Main Street	El Segundo	90245	USA	M

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	20	7	2016-07-05	3
10309	370	3	2016-07-05	1
10310	770	8	2016-07-06	2

You want to create a query that will return all of the customers and all of the orders even if a customer does not have an order.

Which clause should you include in your query?

- ☐ INNER JOIN
- ☐ CROSS JOIN
- ☐ SELF JOIN
- ☐ FULL OUTER JOIN

Answer:

FULL OUTER JOIN

Explanation:

You should create a `FULL OUTER JOIN`. The following query selects all customers and all orders:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders
```

```
ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

A query with the `FULL OUTER JOIN` keyword returns all the rows from the left table, and all the rows from the right table. If there are rows in the left table (`Customers`) that do not have matches in the right table (`Orders`), or if there are rows in the right table (`Orders`) that do not have matches in the left table (`Customers`), the query will list those rows as well.

You should not include a `SELF JOIN` clause. A self join is a join to the existing table. Typically you would see a self join in a scenario in which an `Employee` table contained both workers and supervisors, where employee managers are designated by the `ManagerID` and each manager is also an employee. The self join in the query connects the `Employee` table with itself by using the `EmployeeID` and `ManagerID`.

You should not include a `CROSS JOIN` clause. A cross join is used to join tables that have no common columns. A cross join generates a Cartesian product, which displays all the possible combinations of rows in the joined tables. For each row in one table, a cross join without a `WHERE` clause returns each row in the other table. Thus, the total number of returned rows equals the number of rows in one table multiplied by the number of rows in the other table. SQL statements creating Cartesian products should be used with caution because depending on the number of rows in each of the joined tables, the result set may contain an excessive number of rows. Cartesian products have few useful applications, but are often used to automatically generate a reasonable sample of test data. Because cross joins are used to join tables that have no common columns, using an `ON` clause is not allowed and will generate an error.

You should not include an `INNER JOIN` clause. Inner joins combine information from two tables using a common column. All join operations, except cross joins, must include join criteria, which can be specified in the `ON` clause of the `SELECT` statement. If an inner join is performed, only rows that meet the join criteria are returned from both tables. An inner join returns only rows from the two tables that have a matching value in the common column. An inner join is the default join type.

Item: 13 (Ref:Cert-1Z0-071.6.2.3)
--

Examine the structures of the `product` and `style` tables:

```
product
-----
PRODUCT_ID NUMBER
PRODUCT_NAME VARCHAR2(25)
SUPPLIER_ID NUMBER
QTY_IN_STOCK NUMBER
QTY_ON_ORDER NUMBER
REORDER_LEVEL NUMBER

style
-----
STYLE_ID NUMBER
NAME VARCHAR2(15)
COLOR VARCHAR2(10)
```

You want to create a report displaying all possible `product_id` and `style_id` combinations.

Which three queries could you use? (Choose three.)

- ☐ `SELECT style_id, product_id`
`FROM product`
`CROSS JOIN style`

- ```
ON (style_id = product_id);
```
- ☐ `SELECT style_id, product_id`  
`FROM product`  
`CROSS JOIN style;`
  - ☐ `SELECT style_id, product_id`  
`FROM style`  
`JOIN product`  
`ON style_id = product_id;`
  - ☐ `SELECT style_id, product_id`  
`FROM product`  
`NATURAL JOIN style;`
  - ☐ `SELECT style_id, product_id`  
`FROM style`  
`JOIN product`  
`USING (style_id);`
  - ☐ `SELECT style_id, product_id`  
`FROM style, product;`

Answer:

```
SELECT style_id, product_id
FROM product
CROSS JOIN style;

SELECT style_id, product_id
FROM product
NATURAL JOIN style;

SELECT style_id, product_id
FROM style, product;
```

## Explanation:

To produce the report containing all possible combinations of `product_id` and `style_id`, you could use either the `SELECT` statement that implements a `CROSS JOIN` with no `ON` clause, the `SELECT` statement listing only the table names with no `WHERE` clause, or the statement that implements a `NATURAL JOIN`. Each of these statements will create an intentional Cartesian product, joining all rows in the `product` table to all rows in the `style` table. This will produce a report containing all possible combinations of `product_id` and `style_id` as you desired.

At first glance it may seem that the statement that implements a `NATURAL JOIN` would generate an error. A `NATURAL JOIN` joins the two tables using all columns with the same name. But because these two tables have no columns with the same name, a cross product is produced.

The `SELECT` statement that implements a `CROSS JOIN` including an `ON` clause is incorrect. The `product` and `style` tables have no common column. Therefore, using an `ON` clause will generate an error.

The statement that uses a simple join including the `ON` clause is incorrect because the `style_id` column and the `product_id` column are used to join these tables. All rows whose `style_id` match a `product_id` would be included, and this is not what you desired.

The statement that implements a simple join with a `USING` clause is incorrect. A `USING` clause is used to join two tables on a column with the same name, and these two tables have no common column named `style_id`. When this statement executes, an `ORA-00904: invalid column name error` occurs.

SQL statements creating Cartesian products should be used with caution because, depending on the number of

rows in each of the joined tables, the result set may contain an excessive number of rows. Cartesian products have few useful applications, but are often used to automatically generate a reasonable sample of test data.

|                                          |
|------------------------------------------|
| <b>Item: 14 (Ref:Cert-1Z0-071.6.1.1)</b> |
|------------------------------------------|

You have a script that has the following query:

```
SELECT e1.Lname||' works for '||e2.Lname
 "Workers and Their Supervisors"
FROM employees e1, employees e2
WHERE e1.ManagerId = e2.EmployeeId
 AND e1.Lname LIKE 'T%'
ORDER BY e1.Lname;
```

What type of join is used in the script?

- ☐ Semijoin
- ☐ Antijoin
- ☐ Outer join
- ☐ Self join

Answer:

**Self join**

### Explanation:

The query uses a self join. A self join is a join to the existing table. In this scenario, the employees table contains both workers and supervisors. Employees have their manager designated by the `ManagerId`. Each manager is also an employee. The join in the query connects the `Employees` table with itself by using the `EmployeeId` and `ManagerId`.

The query is not an example of an antijoin. The antijoin returns rows from the left side of the predicate that fail to the subquery on the right side. The antijoin has two or more tables that are joined. The following is an example of an antijoin:

```
SELECT * FROM employees
WHERE DepartmentId NOT IN
 (SELECT DepartmentId FROM Departments
 WHERE OfficeId = 231)
ORDER BY Lname;
```

The query is not an example of a semijoin. A semijoin returns rows that match a subquery that has an `EXISTS` clause in the subquery, but does not duplicate rows from the left side of the predicate when multiple rows on the right side of the predicate satisfy the subquery. The following is an example of a semijoin:

```
SELECT * FROM division
WHERE EXISTS
 (SELECT * FROM employees
 WHERE division.DivisionId= employees.DivisionId
 AND employees.salary> 50000)
```

```
ORDER BY DivisionName;
```

The example is not an example of an outer join. An outer join returns all rows that satisfy the condition of the join, as well as, returning some or all of those rows from one table for which no rows from the other table in the join satisfy the condition of the join. Outer joins can be either left outer joins or right outer joins. A right outer join of tables, `table1` and `table2`, returns all rows of `table2`. A right outer join of tables, `table1` and `table2`, returns all rows of `table1`.