Q1.

Entity Integrity(一致性) constraint：

no primary key value can be NULL. This is because the primary key value is used to identify individual tuples in a relation. Having NULL values for the primary key implies that we cannot identify some tuples. For example, if two or more tuples had NULL for their primary keys, we may not be able to distinguish them if we try to reference them from other relations.

Primary key 代表 entity，所以 primary key 不能為 NULL

Primary key 是代表資料庫每一筆 tuple 的 idntity，所以 primary key 不能有 NULL 因為 domain constraints，key 是 unique 的

The primary key attributes PK of each relation schema R in S cannot have null values in any tuple of r(R).

This is because primary key values are used to identify the individual tuples.

t[PK] ≠ null for any tuple t in r(R)

If PK has several attributes, null is not allowed in any of these attributes

Note: Other attributes of R may be constrained to disallow null values, even though they are not members of the primary key.

foreign key：

reference the primary key attributes PK of the another referenced relation R2.
If a relation schema includes the primary key of another relation schema, that attribute is called the foreign key

Reference 其他 table 的 primary key 的 key

EX：

EMPLOYEE

SSN	SUPERSSN	DNO
e1	e6	2
e3	e4	2
e4	e5	3

WORK_ON

ESSN	PNO	HOURS
e1	p1	5
e3	p1	8
e4	p3	7
e5	p4	6

DEPT

DNumber	Dname	MGRSSN
1	Develop	e21
2	Design	e21
3	AI	e39

WORK_ON's ESSN is a foreign key since it's reference to **EMPLOYEE**'s primary key (SSN)
EMPLOYEE's DNO is a foreign key because it is reference to **DEPT**'s primary key (Dnumber)
DEPT's MGRSSN is a foreign key because it is reference to **EMPLOYEE**'s primary key (SSN)

referential integrity constraint：

foreign key either reference 存在的值 or 是 NULL

specified between two relations and is used to maintain the consistency among tuples in the two relations. Informally, the referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation

A constraint involving two relations

The previous constraints involve a single relation.

Used to specify a relationship among tuples in two relations:

The referencing relation and the referenced relation.

Tuples in the referencing relation R1 have attributes FK (called foreign key attributes) that reference the primary key attributes PK of the referenced relation R2.

A tuple t1 in R1 is said to reference a tuple t2 in R2 if t1[FK] = t2[PK].

A referential integrity constraint can be displayed in a relational database schema as a directed arc from R1.FK to R2.

Statement of the constraint

The value in the foreign key column (or columns) FK of the the referencing relation R1 can be either:

(1) a value of an existing primary key value of a corresponding primary key PK in the referenced relation R2, or

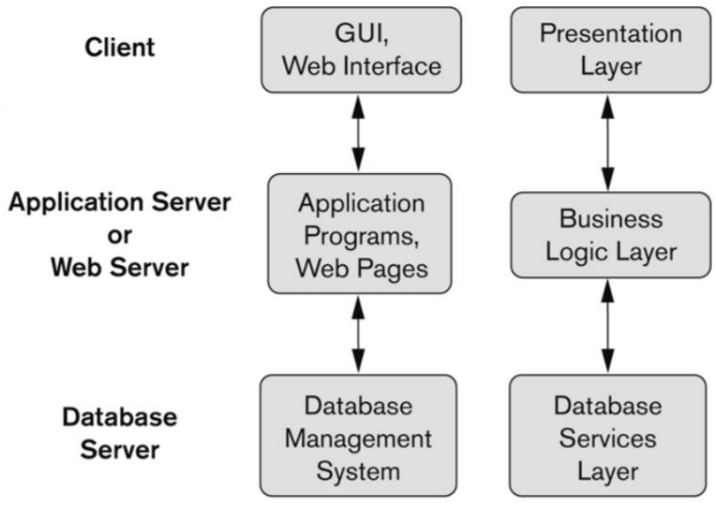
(2) a null.

In case (2), the FK in R1 should not be a part of its own primary key

Q2

(a)

Common for Web applications



Clients

Provide user intervace. Let users can manupulate on it

Receiving user's information and send to Intermediate Layer or presenting information from Intermediate Layer to users

Provide appropriate interfaces through a client software module to access and utilize the various server resources.

Clients may be diskless machines or PCs or Workstations with disks with only the client software installed.

Connected to the servers via some form of a network.

(LAN: local area network, wireless network, etc.)

Represents Web browser, a Java or other application, Applet, WAP phone etc. The client tier makes requests to the Web server who will be serving the request by either returning static content if it is present in the Web server or forwards the request to either Servlet or JSP in the application server for either static or dynamic content.

Intermediate Layer called Application Server or Web Server:

Provide appplication or logic operation

Stores the web connectivity software and the business logic part of the application used to access the corresponding data from the database server

Acts like a conduit for sending partially processed data between the database server and the client.

This layer provides the business services. This tier contains the business logic and the business data. All the business logic like validation of data, calculations, data insertion etc. Are centralized into this tier as opposed to 2-tier systems where the business logic is scattered between the front end and the backend. The benefit of having a centralized business tier is that same business logic can support different types of clients like browser, WAP (Wireless Application Protocol) client, other standalone applications written in Java, C++, C# etc. This acts as an interface between Client layer and Data Access Layer. This layer is also called the intermediary layer helps to make communication faster between client and data layer

Server

Provide access to database

Provides database query and transaction services to the clients

Relational DBMS servers are often called SQL servers, query servers, or transaction servers

Applications running on clients utilize an Application Program Interface (API) to access server databases via standard interface such as:

ODBC: Open Database Connectivity standard

JDBC: for Java programming access

This layer is the external resource such as a database, ERP system, Mainframe system etc. responsible for storing the data. This tier is also known as Data Tier. Data Access Layer contains

methods to connect with database or other data source and to perform insert, update, delete, get data from data source based on our input data

Three-tier Architecture Can Enhance Security:

Database server only accessible via middle tier

Clients cannot directly access database server

Clients contain user interfaces and Web browsers

The client is typically a PC or a mobile device connected to the Web

High performance, lightweight persistent objects.

Scalability – Each tier can scale horizontally.

Performance – Because the Presentation tier can cache requests, network utilization is minimized, and the load is reduced on the Application and Data tiers.

Better Re-usability.

Improve Data Integrity.

Improved Security – Client is not direct access to database.

Forced separation of user interface logic and business logic.

Business logic sits on small number of centralized machines (may be just one).
Easy to maintain, to manage, to scale, loosely coupled etc.

(b)

(i) TRUE AND（FALSE OR UNKNOWN）→ TRUE AND UNKNOWN → UNKNOWN

(ii) (TRUE OR UNKNOWN) AND UNKNOWN → TRUE AND UNKNOWN → UNKNOWN

(iv) (TRUE AND UNKNOWN) OR (UNKNOWN OR UNKNOWN)
→ UNKNOWN OR UNKNOWN → UNKNOWN

Q3

Inherent (implicit) Constraints: These are based on the data model itself. Constraint which are inherent in the data model. (E.g., relational model does not allow a list as a value for any attribute)
Schema-based (explicit) Constraints: Constraint which can be expressed in the schema by using the facilities provided by the model, typically by specifying them in the DDL. (E.g., max. cardinality ratio constraint in the ER model)

Q4

(a)

通常不算 NULL 值

WORKS_ON

Hour
8
10
8
12
NULL
10

Sum: 48

If the collection becomes empty because all values are NULL, SUM will return NULL

(b)

全部都是 empty ➔ Count return 0

如果是用 Agggregate function 且全部都是 NULL ➔ NULL

(c)

COUNT(*) ：

will return the total of all records returned in the result set regardless of NULL values.
counts the number of rows.
Count the number of tuple not of attribute ➔ the tuple would be counted in

WORKS_ON

Hour
8
10
8
12
NULL
10

COUNT (*)： 6

Q5

a

```
UPDATE COMPETITION
SET Score = 'A'
WHERE Song_id IN (SELECT SG.Song_id
                  FROM SONG SG
                  WHERE SG.Language = 'English')
    AND Student_number IN (SELECT ST.Student_number
                           FROM STUDENT ST
                           WHERE ST.Sex = 'male');
```

b

```
INSERT INTO SONG
VALUES (77,'English','Pop','BinMusic');
```

c

```
DELETE FROM COMPETITION
WHERE Student_number IN (SELECT S.Student_number
                         FROM STUDENT S
                         WHERE S.Name = 'Kelly';
```

d

```
SELECT C1.Song_id
FROM COMPETITION C1
WHERE (SELECT COUNT (*)
        FROM COMPETITION C2
        WHERE Score = 'B' AND (SELECT (*)
                                FROM STUDENT
                                WHERE STUDENT.Student_number = C2.Student_number
                                  AND STUDENT.Major = 'MIS') >= 2
                            AND C1.Song_id = C2.Song_id) >= 2;
```

Q6

(1)

```
SELECT FNAME, LNAME, COUNT (*)
FROM EMPLOYEE
GROUP BY SUPERSSN
HAVING COUNT (*) > 3;
```

(2)

```
SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME, COUNT (*)
FROM EMPLOYEE E, EMPLOYEE S, WORKS_ON
WHERE E.SSN = WORKS_ON.ESSN
  AND E.SUPERSSN = S.SSN
  AND NOT EXISTS (SELECT 1
                  FROM DEPARTMENT
                  WHERE DEPARTMENT.MGRSSN = E.SSN)
  AND (SELECT COUNT (*)
       FROM DEPENDENT
       WHERE EMPLOYEE.SSN = DEPENDENT.ESSN) > 2;
GROUP BY WORKS_ON.ESSN;
```

(3)

```
SELECT DEPARTMENT.DNAME, M.SSN, COUNT (*)
FROM DEPARTMENT, EMPLOYEE.E, EMPLOYEE.M
WHERE E.DNO = DEPARTMENT.DNUMBER
  AND E.SEX = 'Male'
  AND (SELECT COUNT (*)
       FROM PROJECT
      WHERE PROJECT.DNUM = DEPARTMENT.DNUMBER) > 5;
GROUP BY E.DNO;
```

(4)

```
SELECT M.FNAME, M.LNAME, DEPARTMENT.DNAME
FROM EMPLOYEE M, DEPARTMENT
WHERE M.DNO = DEPARTMENT.DNUMBER
  AND M.SSN =  DEPARTMENT.MGRSSN
  AND NOT EXTSTS (SELECT *
                  FROM DEPEDENT
                  WHERE DEPEDENT.ESSN= M.SSN)
  AND EXISTS (SELECT *
              FROM EMPLOYMENT E
              WHERE E.DNO = DEPARTMENT.DNUMBER
                 AND M.SALARY < E.SALARY)
```

(5)

```
WITH DNOtwoP (PNO, PCOUNT) AS
     (SELECT WORKS_ON.PNO, COUNT (*)
     FPOM EMPLOYEE , WORKS_ON
     WHERE EMPLOYEE.DNO = 2 AND WORKS_ON.ESSN = EMPLOYEE.SSN
     GROUP BY WORKS_ON.PNO)
SELELT PROJECT.PNAME, DEPARTMENT.DNAME
FROM DEPARTMENT, PROJECT
WHERE PROJECT.DNUM = DEPARTMENT.DNUMBER
  AND (SELECT COUNT (*)
       FROM WORKS_ON
       WHERE PROJECT.PNUMBER = WORKS_ON.PNO)
       > ALL (SELECT PCOUNT
              FROM DNOtwoP);
```

(6)

```
SELECT PROJECT.PNAME, PROJECT.PLOCATION
FROM PROJECT, DEPARTMENT
WHERE PROJECT.DNUM = DEPARTMENT.DNUMBER
   AND DEPARTMENT.DNAME = 'R&D'
   AND (SELECT COUNT (*)
        FROM WORKS_ON
        WHERE WORKS_ON.PNO = PROJECT.PNUMBER) >= 3;
```

(7)

```
SELECT EMPLOYEE.FNAME, EMPLOYEE.LNAME
FROM EMPLOYEE
WHERE NOT EXISTS ((SELECT W1.PNO
                   FROM PROJECT, DEPARTMENT, WORKS_ON W1
                   WHERE PROJECT.PLOCATION = 'Hsinchu'
                       AND PROJECT.DNUM = DEPARTMENT.DNUMBER
                       AND DEPARTMENT.DNAME = 'MIS')
                   EXCEPT (SELECT W2.PNO
                           FROM WORKS_ON W2
                           WHERE W2.PNO = PROJECT.PNUMBER));
```

(8)

```
WITH RECURSIVE SUP_EMP (SupSSN, EmpSSN) AS
        (SELECT SUPERSSN, SSN
        FROM EMPLOYEE
                UNION
        SELECT S.SupSSN, E.SSN
        FROM EMPLOYEE AS E, SUP_EMP AS S
        WHERE E.SUPERSSN = S.EmpSSN)
SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM EMPLOYEE E, SUP_EMP SE, EMPLOYEE S, DEPARTMENT D1
WHERE E.SSN = SE.SupSSN
   AND SE.EmpSSN = S.SSN
   AND E.DNO = D1.DNUMBER
   AND D1.DNAME = 'Research'
   AND NOT EXISTS (SELECT *
                   FROM DEPARTMENT D2
                   WHERE S.SSN = D2.MGRSSN)
   AND (SELECT (*)
        FROM SUP_EMP
        WHERE E.SSN = EmpSSN) > 2;
```

Q7

(a)

(1) $_POST['dept_name']

(2) $_POST['emp_sal']

(3) DNUMBER = DNO

(b)

(4) $_POST['dept_name']

(5) $_POST['emp_sal']

(c)

(6) $q->fetchRow())

(7) $r[0]

(8) $r[1]

(9) $r[2]

Q8

(a)

```
$supervising = ['John' => 'Kevin', 'Darrel' => 'Tim', 'Mary' => 'Jack'];
```

(b)

```
foreach ($supervising as $emp_name => $sup_name)
{
        echo "Employee $emp_name and his/her supervisor $sup_name.\n";
}
```