G51DBS 2008-2009: answers.

1. (a) Give a table returned by

```
SELECT *
FROM Book, Publisher
```

Answer:

```
949 A. Green Memoirs 2001 Thompson 949
949 A. Green Memoirs 2001 Elsevier 287
287 B. Black Biology 2002 Thompson 949
287 B. Black Biology 2002 Elsevier 287
```

(b) Give a table returned by

```
SELECT *
FROM Book CROSS JOIN Publisher
```

Answer: same as in (a)

(c) Give a table returned by

```
SELECT *
FROM Book NATURAL JOIN Publisher
```

Answer:

```
949 A. Green Memoirs 2001 Thompson
287 B. Black Biology 2002 Elsevier
```

(d) Give a table returned by

```
SELECT *
FROM Book INNER JOIN Publisher USING(ISBN)
```

Answer:

```
949 A. Green Memoirs 2001 Thompson 949
287 B. Black Biology 2002 Elsevier 287
```

(e) Rewrite the previous `INNER JOIN` query with `ON` $<$ `condition` $>$ instead of `USING(ISBN)`.

Answer:

```
SELECT *
FROM Book INNER JOIN Publisher
ON Book.ISBN = Publisher.ISBN
```

(f) What are `FULL OUTER JOIN`, `LEFT OUTER JOIN` and `RIGHT OUTER JOIN`? Given an example of the three kinds of outer joins on two tables both of which contain dangles.

Answer: When two tables T1 and T2 are joined on common values in some column(s) C, some tuples in T1 may not have a matching tuple in T2 (with the same value(s) for C) and some tuples in T2 may not have a matching tuple in T1. Such tuples are called *dangles*. Outer joins and like inner joins for matching tuples, but in addition they add dangles to the result table with the values of the missing matching tuple replaced by NULLs. Full outer join does this with both T1 and T2 dangles, left outer join adds T1 dangles and right outer join adds T2 dangles. For example, if T1 is Book and T2 is Publisher, and Book contains a tuple

```
111 Nemo Poems 1900
```

and no tuple with ISBN 111 exists in Publisher, then left outer join and full outer join would contain

```
111 Nemo Poems 1900 NULL NULL
```

Similarly, if Publisher contains

```
Thompson 222
```

and no book with ISBN 222 is in the Book table, then right outer join and full outer join would add a tuple

```
NULL NULL NULL Thompson 222
```

(g) What does it mean for two relations to be union-compatible?

Answer: two relations are union-compatible if they have the same arity and correponding attributes have the same domains of values.

(h) List relational algebra operators and explain what they do.

Answer: relational algebra has union, intersection and difference (all restricted to union-compatible relations), (Extended) Cartesian product, projection and selection. Union, intersection and difference are the same as in set theory. Extended Cartesian product of two relations is a set of all tuples which are formed from some tuple of the first relation continued with some tuple of the second relation. Projection (on attributes A) of relation R is a set of tuples which is obtained from tuples in R by only recording values of attributes in A. Selection (on property P) from relation R is the set of all tuples in R which satisfy P.

(i) Translate the following relational algebra expression into SQL:

$$\pi_{\texttt{Title}}\, \sigma_{\texttt{Year}>2001}\, (\texttt{Book} \times \texttt{Publisher})$$

Answer:

```
SELECT Title
FROM Book, Publisher
WHERE Year > 2001
```

(j) Translate the following SQL query into relational algebra:

```
SELECT Author, Title
FROM Book, Publisher
WHERE Book.ISBN = Publisher.ISBN AND
      PublisherName = 'Elsevier'
```

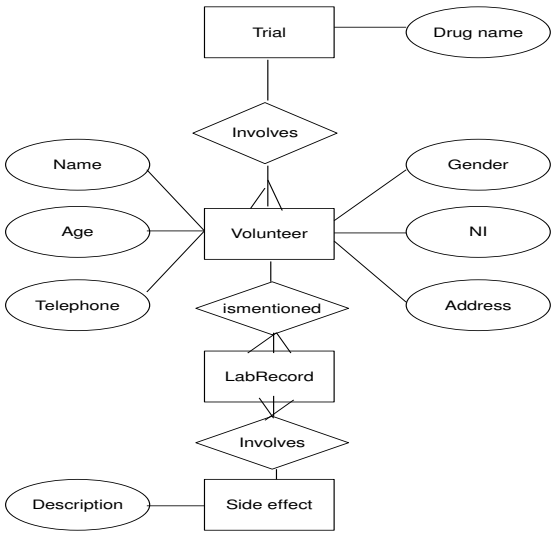
Answer:

$\pi_{\texttt{Author,Title}}\, \sigma_{\texttt{Book.ISBN=Publisher.ISBN AND PublisherName='Elsevier'}}(\texttt{Book} \times \texttt{Publisher})$

2. You are asked to design a database for the following scenario. A research laboratory is running several drug trials on healthy volunteers to check whether drugs have side effects. Each drug has a unique name. Each trial involves exactly one drug and several volunteers (who take the drug and report if they had any side effects). For each volunteer in each trial it needs to be recorded whether the volunteer had any side effects, and if yes, what those side effects were (there could be several side effects experienced by the same person, for example headache, dry mouth, and fever). It is important that side-effects are described using some standard terminology, so that the laboratory can report what proportion of volunteers had the same side effect. For example, the researchers may tell you that headache should always be recorded as 'headache' and not sometimes as 'pain in the head' and sometimes as 'sore head'. There is however no fixed pre-defined set of possible side effects, because new effects can always be discovered (for example the drug may turn people bright green colour). For simplicity, assume that each volunteer takes part in at most one drug trial. Data stored about volunteers is their National Insurance number, name, age, gender, address and telephone number.

   (a) Draw an entity-relationship diagram for the drug trial scenario.

   Possible Answer (other solutions possible, for example where Drug is an entity with a 1:1 relationship to the Trial, will get full marks):



   (b) Write SQL statements to create the tables (don't forget to specify primary and foreign keys). Answer:

```
CREATE TABLE Trial  (
   drugname VARCHAR(50) NOT NULL,
   CONSTRAINT pk_trial  PRIMARY KEY (drugname)
);
CREATE TABLE Volunteer  (
   ni INT NOT NULL,
   name VARCHAR(50) NOT NULL,
   age INT,
   gender CHAR(1),
   address VARCHAR(50),
   telephone  VARCHAR(11),
   drugname VARCHAR(50),
   CONSTRAINT pk_volunteer  PRIMARY KEY (ni),
   CONSTRAINT fk_pTr  FOREIGN KEY (drugname) REFERENCES Trial
);

CREATE TABLE SideEffect (
   description VARCHAR(100) NOT NULL,
   CONSTRAINT pk_sideeffect PRIMARY KEY(description)
);


CREATE TABLE LabRecord (
   description VARCHAR(100) NOT NULL,
   ni INT NOT NULL,
   CONSTRAINT pk_LabRecord PRIMARY KEY(description, ni),
   CONSTRAINT fk_sV1  FOREIGN KEY (description) REFERENCES Sideeffect,
   CONSTRAINT fk_sV2  FOREIGN KEY (ni) REFERENCES Volunteer
);
```

(c) Explain what is referential integrity, and explain it on the example of foreign keys in your answer to question (b).

Answer: Referential integrity requires that a foreign key should either match a candidate key of some tuple in its home relation or be wholly null. So in the tables above, foreign key drugname in Volunteer corresponds to a unique (in fact primary key) column in Trial. If we create a new Volunteer tuple, to begin with it may have NULL for drugname, but once the Volunteer has been assigned to a drug trial, the drug name should match some existing drug name in Trial.

3. This question refers to the following tables: `Children`, `Playgroups`, `Activities`. The `Children` tables contains data about children (names, ages and addresses of parents) - we assume for simplicity that names are unique, `Playgroups` says which child is in which playgroup and `Activities` says what children in the playgroup did on a certain date (for example, went to a zoo).

Children		
Name	Age	Address

Playgroups	
PlaygroupID	Name

Activities		
PlaygroupID	ADate	Description

Write SQL queries to do the following:

(a) Find a list of names of all children.
Answer:

```
SELECT Name
FROM Children
```

(b) Find a list of names of all children aged 4.
Answer:

```
SELECT Name
FROM Children
WHERE Age = 4
```

(c) Return a list of names and addresses of all children in playgroup with ID 1.
Answer:

```
SELECT Name, Address
FROM Children, Playgroups
WHERE Children.Name = Playgroups.Name AND PlaygroupID = 1
```

(Alternatively, can be done with a NATURAL JOIN or a subquery).

(d) Find a list of names and ages of children in a playgroup which went to the zoo on the 21st of February 2009 (`Activities.Description` value is `Zoo`).
Answer:

```
SELECT Name, Age
FROM Children, Playgroups, Activities
WHERE Children.Name = Playgroups.Name AND
```

```
Playgroups.PlaygroupID = Activities.PlaygroupID AND
Activities.Description = 'Zoo' AND
ADate = '21-Feb-2009'
```

(or they can use a natural join, subqueries, whatever.)

(e) Return a list of playgroup IDs and average age of children for each playgroup.
Answer:

```
SELECT PlaygroupID, AVG(Age)
FROM Children, Playgroups
WHERE Children.Name = Playgroups.Name
GROUP BY PlaygroupID
```

(f) Find names and addresses of all children who are in the same playgroup as a child called 'Amy Jones'.
Answer:

```
SELECT Name, Address
FROM Children, Playgroups
WHERE Children.Name = Playgroups.Name AND PlaygroupID IN
     (SELECT PlaygroupID
      FROM Playgroups
      WHERE Name = 'Amy Jones')
```

Or they can do

```
SELECT A.Name
FROM Playgroups A, Playgroups B
WHERE A.PlaygroupID = B.PlaygroupID AND
      B.Name = 'Amy Jones'
```

(g) Add an extra column to the `Activities` table which is called `Supervisor` and has the type of values `VARCHAR(20)`.
Answer:

```
ALTER TABLE Activities
ADD Supervisor
VARCHAR(20)
```

[ADD COLUMN as opposed to ADD is allowed by SQL but not by Oracle SQL - allowed in the answer].

(h) Replace description `Zoo` in the `Activities` table everywhere with `Visit to the Zoo`.
Answer:

```
UPDATE Activities
SET Description = 'Visit to the Zoo'
WHERE Description = 'Zoo'
```

4. (a) Define functional dependency.

   Answer: A relation R has a functional dependency X → Y if for every two tuples s and t in R, if s and t have the same values for the attributes in X, then they have the same values for the attributes in Y.

   (b) Define 2 NF.

   Answer: A relation is in 2NF if there is no functional dependency X → Y where Y is a non-key attribute and X is a strict subset of a candidate key.

   (c) Define 3 NF.

   Answer: A relation is in 3NF if there are no non-trivial functional dependencies X → Y, Y → Z where X is a candidate key, and Z is a non-key attribute.

   (d) Define BCNF.

   Answer: A relation is in BCNF, if for every non-trivial functional dependency X → Y, X is a (super) key.

   (e) What are insertion, deletion and update anomalies?

   Answer: Insertion anomaly is a situation when it is impossible to insert information about some attributes X because this would make part of a primary key to be null, violating entity integrity. Deletion anomaly is when it is impossible to delete information about some objects without removing unrelated information. Update anomaly arises when in order to change a value of some attribute we need to update many tuples.

   (f) Consider a relation Listing with attributes Cinema, Film, Day, Time, Certificate. Sample tuples:

   ```
   Savoy, 'Slumdog Millionaire', Wed, 18:00, 15
   Savoy, 'Slumdog Millionaire', Wed, 20:00, 15
   Cineworld, 'Slumdog Millionaire', Wed, 20:30, 15
   Cineworld, 'Vicky Christina Barcelona', Wed, 20:30, 12A
   ```

   Each film is assigned a certificate by the British Board of Film Classification; certificate 15 means that nobody younger than 15 can see this film in a cinema. The same cinema can show a film on multiple times during a day, and may show different films at the same time (on different screens).

   Does this relation violate the second normal form requirements? Explain your answer.

   Answer: a non-key attribute Certificate is determined by Film, which is part of a candidate key (Cinema, Film, Day, Time).

   (g) Give an example of update anomalies in this relation.

If the certficate for a film were to be changed (say Slumdog Millionaire was downgraded to 12A or upgraded to 18), each tuple which contains this film would have to be updated, for every cinema, day, and showing time.

(h) Decompose this relation into BCNF, and explain why the resulting relations are in BCNF.

Decomposition:

Showing (Cinema, Film, Day, Time)

and Film (Film, Certificate).

Both are in BCNF because in the Showing relation, no subset of the attributes determines any other attribute, and the only non-trivial dependency in Film is from Film to Certificate.

5. (a) What is a NULL? Explain A-marks and I-marks.

   Answer: NULLS are markers for missing values. A-marks are NULLs which replace applicable but unknown information; I-marks replace inapplicable information.

   (b) How does SQL evaluate conditions for tuples containing NULLS? Explain the general approach.

   For a tuple {`Name` : `John`, `Age` : `NULL`}, will the following `WHERE` conditions evaluate to true?

```
1. WHERE Age > 18
2. WHERE Age > 18 OR Name = 'John'
3. WHERE Age > 18 OR NOT (Age > 18)
```

   Answer: SQL essentially uses 3-valued logic, where comparisons involving NULLs evaluate to a third value (unkown or 1/2, while false is 0 and true is 1). NOT(x) = 1-x, OR(x,y) = max(x,y), and AND(x,y) = min(x,y). Only the tuples for which the condition evaluates to true are returned. Only the second condition above evaluates to 'true', the rest to 'unknown'.

   (c) How are arithmetic $(+, -, *)$ operations applied to NULLs evaluated in SQL?

   Answer: the result is always NULL.

   (d) How are aggregate operations (SUM, AVG, Count) on NULLs evaluated in SQL?

   NULLs are not included in SUM, AVG and Count (for the column).

   (e) In response to what problem was the timestamping protocol developed? In order to avoid race hazard in concurrent access to a database.

   (f) Describe the timestamping protocol. Answer:

   Each transaction T is given a different timestamp $TS(T)$ (usually given by system time when transaction starts); transactions which start earlier have a smaller timestamp.

   Each resource X is given two timestamps, $R(X)$ which is the largest timestamp of any transaction that has read X, and $W(X)$ the largest timestamp of any transaction that has written X.

   If T tried to read X and $TS(T) < W(X)$, T is rolled back and restarted with a new timestamp. Otherwise read succeeds and $R(X)$ becomes $max(TS(T), R(X))$. If T tries to write X and $TS(T) < W(X)$ or $TS(T) < R(X)$, T is rolled back and restarted with a new timestamp. Otherwise write succeeds and $W(X)$ is set to be $TS(T)$.

(g) Trace the timestamping protocol for the following two transactions T1 and T2, assuming that the statements are executed in a strictly alternating way (first statement of T1 followed by the first statement of T2 followed by the second statement of T1, and so on) and there are no other transactions. At each step, indicate what the time stamps of T1 and T2 are, and what the read and write timestamps of resources X, Y are. Assume that before T1 and T2 are executed the timestamps of resources are 0. Trace until both transactions can commit.

```
Transaction 1                    Transaction 2

Write(X)                          Read(Y)
Write(Y)                          Write(X)
```

Answer:

```
start: TS(T1) = 1, TS(T2) = 2, R(X)=W(X)=R(Y)=W(Y)=0.

T1: Write(X)

W(X) = 1, the rest is as before: R(X)=R(Y)=W(Y)=0.

T2: Read(Y)

R(Y) = 2, the rest is as before: W(X) = 1,  R(X)=W(Y)=0.

T1: Write(Y)

TS(T1) < R(Y), so this does not succeed and T1 is rolled back and
restarted with a new timestamp. TS(T1) = 3.

T2: Write(X)

W(X) = 2, R(Y) = 2, R(X)=W(Y)=0.
T2 commits.

T1: Write(X)
W(X)=3, R(Y) = 2, R(X)=W(Y)=0.

T1: Write(Y)
W(X)=3, R(Y) = 2, W(Y) = 3, R(X)=0.
T1 commits.
```