CS460 Spring, 2016

Homework #4
TA: Shuo Yang

---

1. 5.12 (d)
   RA:

$$\pi_{price,type}(Room \bowtie_{Room.hotelNo=Hotel.hotelNo} (\sigma_{hotelName=\text{“}GrosvenorHotel''}(Hotel)))$$

   TRC:

$$\{R.price, R.type | Room(R) \wedge (\exists H)(Hotel(H) \wedge (R.hotelNo = H.hotelNo) \wedge (H.hotelName = \text{“}GrosvenorHotel''))\}$$

   DRC:

$$\{price, type | (\exists roomNo, hotelNo, hotelNo1, hotelName, city) (Room(roomNo, hotelNo, type, price) \wedge Hotel(hotelNo1, hotelName, city) \wedge (hotelNo = hotelNo1) \wedge (hotelName = \text{“}GrosvenorHotel''))\}$$

2. 5.26
   RA:

$$\pi_{title}(\sigma_{year=2012}(Book))$$

   TRC:

$$\{B.title | Book(B) \wedge B.year = 2012\}$$

   DRC:

$$\{title | (\exists ISBN, edition, year)(Book(ISBN, title, edition, year) \wedge year = 2012)\}$$

3. 5.28
   RA:

$$\pi_{copyNo}(\sigma_{title=\text{“}LoadoftheRings''}(Book) \bowtie_{ISBN} (\sigma_{available='Y'}(BookCopy)))$$

   TRC:

$$\{BC.copyNo | BookCopy(BC) \wedge (\exists B)(Book(B) \wedge (B.ISBN = BC.ISBN) \wedge (BC.available =' Y') \wedge (B.title = \text{“}LordoftheRings''))\}$$

   DRC:

$$\{copyNo | (\exists ISBN, title, edition, year, ISBN, available)(Book(ISBN, title, edition, year) \wedge BookCopy(copyNo, ISBN, available) \wedge available =' Y' \wedge title = \text{“}LordoftheRings'')\}$$

4. 6.4
   An aggregate function can be used only in the SELECT list and in the HAVING clause.

   Apart from COUNT(*), each function eliminates nulls first and operates only on the remaining non-null values. COUNT(*) counts all the rows of a table, regardless of whether nulls or duplicate values occur.

5. 6.10
   SELECT * FROM Room WHERE price < 40 AND type IN ('D', 'F') ORDER BY price;

   (Note, ASC is the default setting for ORDER BY).

6. 6.14
   SELECT SUM(price) FROM Room WHERE type = 'D';

7. 6.20
   SELECT * FROM Room r
   WHERE roomNo NOT IN
   (SELECT roomNo FROM Booking b, Hotel h
   WHERE (dateFrom <= CURRENT_DATE AND
   dateTo >= CURRENT_DATE) AND
   b.hotelNo = h.hotelNo AND hotelName = "Grosvenor Hotel");

8. 6.25
   SELECT MAX(X)
   FROM (SELECT type, COUNT(type) AS X
   FROM Booking b, Hotel h, Room r
   WHERE r.roomNo = b.roomNo AND b.hotelNo = h.hotelNo AND
   city = "London" GROUP BY type);

9. 7.7
   Materialized view is a temporary table that is stored in the database to represent a view, which is maintained as the base table(s) are updated.

   Advantages:
   - may be faster than trying to perform view resolution.
   - may also be useful for integrity checking and query optimization.

10. 7.11 (b,d,f)
    CREATE DOMAIN RoomPrice AS DECIMAL(5, 2)
    CHECK(VALUE BETWEEN 10 AND 100);

    CREATE TABLE Room(
        roomNo integer NOT NULL,
        hotelNo integer NOT NULL,
        type char(1) NOT NULL,
        price RoomPrice NOT NULL,
        PRIMARY KEY (roomNo, hotelNo),
        FOREIGN KEY (hotelNo) REFERENCES Hotel
        ON DELETE CASCADE ON UPDATE CASCADE
    );

    CREATE TABLE Booking(
        hotelNo integer NOT NULL,
        guestNo integer NOT NULL,
        dateFrom date NOT NULL,
        dateTo date NULL,
        roomNo integer NOT NULL,
        PRIMARY KEY (hotelNo, guestNo, dateFrom),
        FOREIGN KEY (hotelNo) REFERENCES Hotel
        ON DELETE CASCADE ON UPDATE CASCADE,
        FOREIGN KEY (guestNo) REFERENCES Guest
        ON DELETE NO ACTION ON UPDATE CASCADE,

```
    FOREIGN KEY (roomNo) REFERENCES Room
    ON DELETE NO ACTION ON UPDATE CASCADE,
    CONSTRAINT GuestBooked
    CHECK (NOT EXISTS (SELECT *
                       FROM Booking b
                       WHERE b.dateTo > Booking.dateFrom AND
                       b.dateFrom < Booking.dateTo AND
                       b.guestNo = Booking.guestNo)));
```

11. 7.14

```
CREATE VIEW BookingOutToday AS
    SELECT g.guestNo, g.guestName, g.guestAddress, r.price*(b.dateTo-b.dateFrom)
    FROM Guest g, Booking b, Hotel h, Room r
    WHERE g.guestNo = b.guestNo AND r.roomNo = b.roomNo AND
    b.hotelNo = h.hotelNo AND h.hotelName = "Grosvenor Hotel" AND
    b.dateTo = CURRENT_DATE;
```

12. 8.5

PL/SQL uses **cursors** to allow the rows of a query result to be accessed one at a time. In effect, the cursor acts as a pointer to a particular row of the query result. The cursor can be advanced by 1 to access the next row. A cursor must be declared and opened before it can be used, and it must be closed to deactivate it after it is no longer required. Once the cursor has been opened, the rows of the query result can be retrieved one at a time using a FETCH statement, as opposed to a SELECT statement.

13. 8.7

The BEFORE keyword indicates that the trigger should be executed before an insert is applied. It could be used to prevent a member of staff from managing more than 100 properties at the same time.

The AFTER keyword indicates that the trigger should be executed after the database table is updated. It could be used to create an audit record.

14. 8.8

There are two types of trigger: row-level triggers (FOR EACH ROW) that execute for each row of the table that is affected by the triggering event, and statement-level triggers (FOR EACH STATEMENT) that execute only once even if multiple rows are affected by the triggering event. An example of a row-level trigger is shown in Example 8.2 where we create an AFTER row-level trigger to keep an audit trail of all rows inserted into the Staff table. An example of a statement-level trigger to set a new sequence number for an update is seen in the middle of Figure 8.5(b).

15. 8.11 (b)

```
CREATE TRIGGER RoomPrice
BEFORE INSERT ON Room
FOR EACH ROW
WHEN (new.type = 'D')
DECLARE vMaxSingleRoomPrice NUMBER;
BEGIN
    SELECT MAX(price) INTO vMaxSingleRoomPrice
    FROM Room
    WHERE type = S;
    IF (new.price < vMaxSingleRoomPrice)
        raise_application_error(-20000, "Double room price must be higher than higest single room price" ||
vMaxSingleRoomPrice);
    END IF;
END;
```

16. 14.4

Functional dependency describes the relationship between attributes in a relation. For example, if A and B

are attributes of relation R, B is functionally dependent on A (denoted A $\rightarrow$ B), if each value of A in R is associated with exactly one value of B in R.

Functional dependency is a property of the meaning or semantics of the attributes in a relation. The semantics indicate how the attributes relate to one another and specify the functional dependencies between attributes. When a functional dependency is present, the dependency is specified as a constraint between the attributes.

17. 14.6
Identifying all functional dependencies between a set of attributes should be relatively simple if the meaning of each attribute and the relationships between the attributes are well understood. This type of information may be provided by the enterprise in the form of discussions with users and/or appropriate documentation such as the users requirements specification. However, if the users are unavailable for consultation and/or the documentation is incomplete then depending on the database application it may be necessary for the database designer to use their common sense and/or experience to provide the missing information. As a contrast to this, consider the situation where functional dependencies are to be identified in the absence of appropriate information about the meaning of attributes and their relationships. In this case, it may be possible to identify functional dependencies if sample data is available that is a true representation of all possible data values that the database may hold.

18. 14.7
A table in unnormalized form contains one or more repeating groups. To convert to first normal form (1NF) either remove the repeating group to a new relation along with a copy of the original key attribute(s), or remove the repeating group by entering appropriate data in the empty columns of rows containing the repeating data.

19. Given $C \rightarrow BDEF$ and $D \rightarrow AG$, by the decomposition rule, we know that:

$$C \rightarrow B$$
$$C \rightarrow D$$
$$C \rightarrow E$$
$$C \rightarrow F$$
$$D \rightarrow A$$
$$D \rightarrow G$$

By the transitivity rule, since $C \rightarrow D$ and $D \rightarrow A, D \rightarrow G$, we know that:

$$C \rightarrow A$$
$$C \rightarrow G$$

By relexivity rule, we know that $C \rightarrow C$. Thus:

$$C^+ = ABCDEFG$$

By the given set of FDs, no attribute functionally dependent on A except itself, thus:

$$A^+ = A$$

20. 1) Put the FDs in a standard form:

$$W \rightarrow Z$$
$$WY \rightarrow X$$
$$Y \rightarrow Z$$
$$W \rightarrow X \text{ [decomposition rule]}$$
$$W \rightarrow Y \text{ [decomposition rule]}$$

2) Minimize the left side of each FD:
we can replace $WY \to X$ with $W \to X$ since $W \to Y$:

$$W \to Z$$
$$W \to X$$
$$Y \to Z$$
$$W \to X$$
$$W \to Y$$

3) Delete redundant FDs:
one of $W \to X$ is redundant. And $W \to Z$ is redundant because it can be inferred by $W \to Y$ and $Y \to Z$.
Thus, the minimal cover of S is:

$$W \to X$$
$$Y \to Z$$
$$W \to Y$$