# CS2102: Database Systems (AY2022-2023 – Sem 2)

# Final Exam

## Instructions

1. Please read **ALL** instructions carefully.
2. This assessment contains **SIXTEEN (16)** questions including subquestions:
   (a) There are 11 Multiple Response Questions (MRQ):
   1.2 - 1.4, 2.2, 3.1b, 3.2b, 4.1(a-d) - 4.2
   (b) There is 1 Fill in the Blank Questions (FITB): 1.1
   (c) There are 2 Multiple Choice Question (MCQ): 2.1, 4.3,
   (d) There are 2 True/False Question: 3.1a, 3.2a
3. There is a total of **EIGHTY (80)** points for this assessment.
4. Answer **ALL** questions.
5. All the assessment is be done using Examplify:
   (a) This is a secure assessment (no internet ; no other software)
   (b) This is a closed-book exam ; you are allowed 1 A4 cheat sheet double-sided
   (c) The choices on Examplify may be in a different order
6. No additional time will be given to submit.
7. Use the question number shown on Examplify when asking question.
   - If the answer is clear from the question pdf/Examplify, we will reply with "No Comment".
8. Failure to follow each of the instructions above may result in deduction of your marks.

## Good Luck!

# 1 SQL

We want to create a database for room booking. Our database should contain:

- **Employees**
  - Identified by employee id (`INT`).
  - Must store the name of the employee (`TEXT`).
- **Rooms**
  - Identified by floor number (`INT`) and room number (`INT`).
  - Must store the name of the room (`TEXT`).

A room can be booked by employees following the constraints below.

- When a room is booked, it is booked for the whole day.
- A room can be booked by at most one employee on any single day. In other words, no two different employees can book the same room on the same day. However, it is possible that the room is not booked by any employee.
- An employee can book at most one room on any single day. In other words, no two different rooms can be booked by the same employee on the same day. However, it is possible that the employee does not book any room.

**1.1** (14 points) **CREATE TABLE.** *This questions is split into two on Examplify*

Your answer for each blank should only be exactly **one** simple constraint (*e.g., `UNIQUE`, etc*) which cannot be a `CHECK` constraint or an attribute. You should not add unnecessary constraints. Unnecessary constraints includes constraints not specified above as well as constraints already enforced in other parts (*e.g., `NOT NULL` or `UNIQUE` on an attribute with `PRIMARY KEY` constraint*). If you feel that there should be nothing inside the box (*e.g., because it will only be unnecessary constraints*), simply put in "-" (*without the quote*). An empty box is treated as *unanswered*. Lastly, in some cases, the order may not matter (*e.g, some primary key or foreign key attributes ordering*).

```
CREATE TABLE Employees (
  eid    INT  [ PRIMARY KEY ] ,
  ename  TEXT [ NOT NULL ]
);

CREATE TABLE Rooms (
  rfloor INT  [ - ] ,
  rnum   INT  [ - ] ,
  rname  TEXT [ NOT NULL ] ,
  PRIMARY KEY ( [ rnum ] , [ rfloor ] )
);
```

```
CREATE TABLE Bookings (
  bfloor INT   [ - ]        ,
  bnum   INT   [ - ]        ,
  beid   INT   [ NOT NULL ] ,
  bday   DATE  [ - ]        ,
  PRIMARY KEY ( [bnum], [bfloor], [bday] ),
  UNIQUE ( [beid], [bday] ),
  FOREIGN KEY ( [bnum], [bfloor] ) REFERENCES Rooms(rnum, rfloor),
  FOREIGN KEY ( [beid] ) REFERENCES Employees
);
```

**NOTES:**

Primary key of `Rooms` has to be in that specific order because of foreign key of `Bookings` referencing that specific order. Primary key of `Bookings` as well as its unique constraints can be in any order.

`beid` of `Bookings` (*and only this from* ***`Bookings`***) has to have `NOT NULL` because we require candidate keys to be enforced additionally with `NOT NULL`. Note that `beid` is in `UNIQUE` constraints because this is a candidate key with 2 attributes. `bday` of `Bookings` should not have `NOT NULL` because it is already inside a `PRIMARY KEY` constraint.

**1.2** (6 points) **INSERT INTO.** Assume that the following insert statements have been performed successfully.

```
INSERT INTO Employees VALUES (1, 'A');
INSERT INTO Employees VALUES (2, 'B');
INSERT INTO Employees VALUES (3, 'C');
INSERT INTO Rooms VALUES (1, 1, 'R1');
INSERT INTO Rooms VALUES (1, 2, 'R2');
INSERT INTO Rooms VALUES (2, 1, 'R3');
```

Select ALL insert statement that will **NOT** be rejected. For choices with two insert statements, select only if none are rejected. Consider each choices independently and assumes all the constraints are enforced correctly. Note that dates can be inserted as a string in the format `'YYYY-MM-DD'`, where `YYYY` is the year, `MM` is the month, and `DD` is the date.

(✓) `INSERT INTO Bookings VALUES (1, 2, 3, '2023-04-29');`

(B) `INSERT INTO Bookings VALUES (2, 2, 2, '2023-04-29');`

(C) `INSERT INTO Bookings VALUES (3, 2, 1, '2023-04-29');`

(✓) `INSERT INTO Bookings VALUES (1, 1, 1, '2023-04-29');`

(E) `INSERT INTO Bookings VALUES (1, 2, 1, '2023-04-29');`
`INSERT INTO Bookings VALUES (2, 1, 1, '2023-04-29');`

(✓) `INSERT INTO Bookings VALUES (1, 2, 1, '2023-04-29');`
`INSERT INTO Bookings VALUES (2, 1, 2, '2023-04-29');`

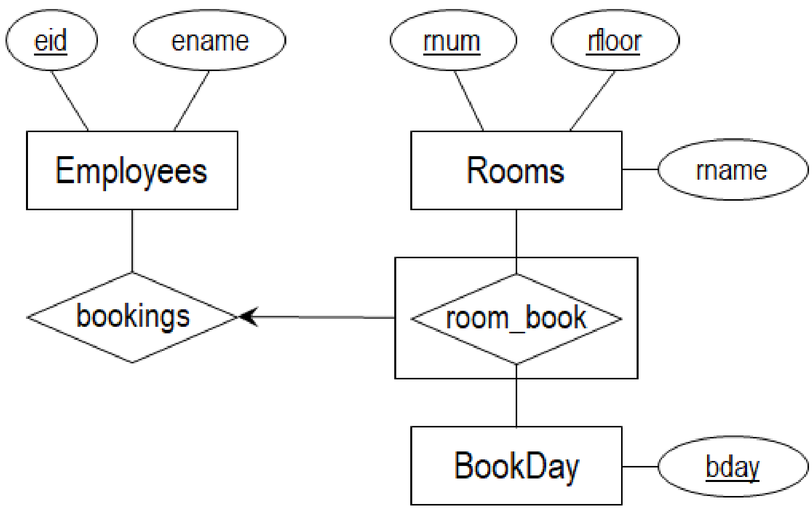
(G) `INSERT INTO Bookings VALUES (2, 2, 2, '2023-04-29');`
`INSERT INTO Bookings VALUES (2, 2, 1, '2023-04-29');`

(H) `INSERT INTO Bookings VALUES (1, 2, 3, '2023-04-29');`
`INSERT INTO Bookings VALUES (3, 2, 1, '2023-04-29');`

(I) *None of the above*

> **NOTES:**
> This does not depend on the order of `PRIMARY KEY` but only on the order of attributes in the `CREATE TABLE`. So, for `Bookings` specifically, we have the following order of attributes `(bfloor, bnum, beid, bday)`.

**1.3** (5 points) **ER Diagram.** Now consider the following ER diagram representation.




Select ALL constraints that are **NOT** enforced by the ER diagram.

(A) Employees are identified by their employee id.

(B) Rooms are identified by floor number and room number.

(C) No two different employees can book the same room on the same day.

(D) It is possible that a room is not booked by any employee.

(✓) No two different rooms can be booked by the same employee on the same day.

(F) It is possible that an employee does not book any room.

(G) *None of the above*

**NOTES:**

For some, it is probably easier to see if you look at the functional dependencies.

(A) $\{$`eid`$\} \rightarrow \{$`ename`$\}$

(B) $\{$`rnum, rfloor`$\} \rightarrow \{$`rname`$\}$

(C) $\{$`rnum, rfloor, bday`$\} \rightarrow \{$`eid`$\}$
(*so now, you cannot have two different employees*)

(D) This is because there is no total participation constraints of room_book with respect to bookings (*as well as Rooms with respect to room_book*).

(✓) $\{$`eid, bday`$\} \not\rightarrow \{$`rnum, rfloor`$\}$
(*so now, you can have two different rooms!*)

(F) This is because there is no total participation constraints of Employees with respect to bookings.

**1.4** (5 points) **SQL Query.** Consider the following instance of table `Bookings`.

Bookings

| bfloor | bnum | beid | bday |
|---|---|---|---|
| 1 | 1 | 1 | 2023-04-29 |
| 1 | 3 | 3 | 2023-04-29 |
| 1 | 4 | 2 | 2023-04-29 |
| 1 | 2 | 1 | 2023-04-29 |
| 2 | 2 | 2 | 2023-04-29 |
| 1 | 1 | 3 | 2023-04-28 |
| 2 | 1 | 3 | 2023-04-28 |
| 2 | 2 | 3 | 2023-04-28 |
| 2 | 3 | 2 | 2023-04-28 |
| 1 | 2 | 1 | 2023-04-27 |
| 2 | 1 | 2 | 2023-04-27 |
| 1 | 1 | 3 | 2023-04-27 |

Select ALL the rows that will appear in the result if we run the following query.

```
SELECT bfloor, COUNT(DISTINCT beid)
FROM   Bookings
WHERE  bday > '2023-04-27'
GROUP BY bday, bfloor;
```

(✓) | 1 | 1 |    (✓) | 1 | 3 |    (✓) | 2 | 1 |    (G) | 2 | 3 |

(B) | 1 | 2 |    (D) | 1 | 4 |    (✓) | 2 | 2 |    (H) | 2 | 4 |

(I) *None of the above*

**NOTES:**

Firstly, while PostgreSQL can compare DATE in a logical way, even comparing this as string is sufficient given the way our dates are written. So exclude the last three rows.

Then, group this by `bday` and `bfloor`, you get the following grouping with `bday` moved next to `bfloor`

`Bookings`

| `bfloor` | `bday` | `bnum` | `beid` |
|---|---|---|---|
| 1 | 2023-04-29 | 1 | 1 |
| 1 | 2023-04-29 | 3 | 3 |
| 1 | 2023-04-29 | 4 | 2 |
| 1 | 2023-04-29 | 2 | 1 |
| 2 | 2023-04-29 | 2 | 2 |
| 1 | 2023-04-28 | 1 | 3 |
| 2 | 2023-04-28 | 1 | 3 |
| 2 | 2023-04-28 | 2 | 3 |
| 2 | 2023-04-28 | 3 | 2 |

Lastly, simply compute the number of ***distinct*** `beid` for each grouping. This gives us:

`Bookings`

| `bfloor` | `count` |
|---|---|
| 1 | 3 |
| 2 | 1 |
| 1 | 1 |
| 2 | 2 |

## 2 Stored Procedures

Consider the following `Scores` table. The data types of `sid` and `name` are `TEXT` while the data type of `score` is `INT`.

Scores

| sid | name | score |
|---|---|---|
| s1 | Alice | 71 |
| s2 | Bob | 78 |
| s3 | Cathy | 84 |
| s4 | David | 89 |
| s5 | Eric | 93 |

**2.1** (6 points) Consider the `test_func` function below.

```
CREATE OR REPLACE FUNCTION test_func()
RETURNS INT AS $func$
DECLARE
  curs CURSOR for (SELECT * FROM Scores ORDER BY score desc);
  r RECORD;
  score1 INT;
  score2 INT;
  gap1 INT;
  gap2 INT;
BEGIN
  score1 := -1;
  gap2 := -1;

  OPEN curs;
  LOOP
    FETCH curs INTO r;
    EXIT WHEN NOT FOUND;

    IF (score1 = -1) THEN
      score1 := r.score;
    ELSE
      gap1 := score1 - r.score;
      IF (gap1 > gap2) THEN
        gap2 := gap1;
        score2 := score1;
      END IF;
      score1 := r.score;
    END IF;
  END LOOP;
  CLOSE curs;
  // continue on the next page
```

```
  IF (gap2 > -1) THEN
    RETURN score2;
  ELSE
    RETURN -1;
  END IF;
END;
$func$ LANGUAGE plpgsql;
```

Suppose that we execute the following query. What will be the query result?

```
SELECT * FROM test_func();
```

(A) 4 (C) 6 (E) 8 (✓) 78 (I) 89

(B) 5 (D) 7 (F) 71 (H) 84 (J) 93

(K) *None of the above*

**NOTES:**

The function is finding the largest gap when sorted in descending order.

Scores

| sid | name | score | gap |
|---|---|---|---|
| s5 | Eric | 93 | 4 |
| s4 | David | 89 | 5 |
| s3 | Cathy | 84 | 6 |
| s2 | Bob | 78 | 7 |
| s1 | Alice | 71 | - |

But it does not return the gap, it returns the larger score from the gap. Notice that the gap is between 78 and 71. So it returns 78.

3 marks when answering either 71 or 84.

**2.2** (6 points) Consider the `scores_check_func` function below.

```
CREATE OR REPLACE FUNCTION scores_check_func()
RETURNS TRIGGER AS $func$
DECLARE
  fail_cnt INT;
BEGIN
  SELECT COUNT(*) INTO fail_cnt
  FROM Scores
  where score < 60;

  IF (fail_cnt > 3) THEN
```

```
    RAISE EXCEPTION 'You failed too many students!';
  END IF;

  RETURN NEW;
END;
$func$ LANGUAGE plpgsql;
```

Suppose that we create a trigger `scores_check_trigger` based on the above function, and the execute the following transaction.

```
BEGIN
  INSERT INTO Scores values ('s6', 'Fred', 59);
  INSERT INTO Scores values ('s7', 'Gigi', 58);
  INSERT INTO Scores values ('s8', 'Helen', 57);
  INSERT INTO Scores values ('s9', 'Ivan', 56);
COMMIT;
```

Assumes that `Scores` has **nine (9)** tuples after the transaction is executed. In that case, which of the following definitions of `scores_check_trigger` are possible?

(✓)
```
CREATE TRIGGER scores_check_trigger
BEFORE INSERT ON Scores
FOR EACH ROW
EXECUTE FUNCTION scores_check_func();
```

(B)
```
CREATE TRIGGER scores_check_trigger
AFTER INSERT ON Scores
FOR EACH ROW
EXECUTE FUNCTION scores_check_func();
```

(C)
```
CREATE CONSTRAINT TRIGGER scores_check_trigger
BEFORE INSERT ON Scores
DEFERRABLE INITIALLY IMMEDIATE
FOR EACH ROW
EXECUTE FUNCTION scores_check_func();
```

(D)
```
CREATE CONSTRAINT TRIGGER scores_check_trigger
AFTER INSERT ON Scores
DEFERRABLE INITIALLY IMMEDIATE
FOR EACH ROW
EXECUTE FUNCTION scores_check_func();
```

(E)
```
CREATE CONSTRAINT TRIGGER scores_check_trigger
BEFORE INSERT ON Scores
DEFERRABLE INITIALLY DEFERRED
FOR EACH ROW
EXECUTE FUNCTION scores_check_func();
```

(F)
```
CREATE CONSTRAINT TRIGGER scores_check_trigger
AFTER INSERT ON Scores
DEFERRABLE INITIALLY DEFERRED
```

```
FOR EACH ROW
EXECUTE FUNCTION scores_check_func();
```

(G) *None of the above*

**NOTES:**

Deferrable constraints can only be used on `AFTER` trigger. Since deferrable `BEFORE` trigger is not even a possible definition, it cannot be part of the answer. However, we do need to have a `BEFORE` trigger and, in fact, the only `BEFORE` trigger that we have is the only answer.

# 3 Functional Dependencies

**3.1** (6 points) Consider the following relation $R(A, B, C, D, E, F)$ with the following set of functional dependencies

$$\Sigma = \{\{F\} \to \{B\}, \{D\} \to \{A\}, \{C\} \to \{F\}, \{E, F\} \to \{C\}, \\ \{A, F\} \to \{E\}, \{A, B\} \to \{F\}\}$$

Suppose that we decompose $R$ into $R1(A, B, C, D)$ and $R2(C, D, E, F)$.

(a) (2 points) Is this a lossless-join decomposition?

(✓) True

(B) False

(b) (4 points) Is this a dependency-preserving decomposition? If not, please select all FDs that are **NOT** preserved.

(✓) $\{F\} \to \{B\}$

(B) $\{D\} \to \{A\}$

(C) $\{E, F\} \to \{C\}$

(✓) $\{A, F\} \to \{E\}$

(E) $\{A, F\} \to \{B\}$

(F) $\{A, B\} \to \{F\}$

(G) *None of the above*

**3.2** (7 points) Consider the following relation $R(A, B, C, D, E, F)$ with the following set of functional dependencies

$$\Sigma = \{\{D\} \to \{E\}, \{C\} \to \{A\}, \{C, F\} \to \{E\}, \{A, F\} \to \{C\}, \\ \{C, E\} \to \{D\}, \{B, E\} \to \{A\}, \{A, D\} \to \{B\}, \{B, C\} \to \{F\}\}$$

Suppose that we decompose $R$ into $R1(A, B, C, D)$ and $R2(C, D, E, F)$.

(a) (2 points) Is this a lossless-join decomposition?

(✓) True

(B) False

(b) (5 points) Is this a dependency-preserving decomposition? If not, please select all FDs that are **NOT** preserved.

(A) $\{D\} \to \{E\}$

(B) $\{C\} \to \{A\}$

(C) $\{C, F\} \to \{E\}$

(✓) $\{A, F\} \to \{C\}$

(E) $\{C, E\} \to \{D\}$

(✓) $\{B, E\} \to \{A\}$

(G) $\{A, D\} \to \{B\}$

(H) $\{B, C\} \to \{F\}$

(I) *None of the above*

# 4 Normal Forms

**4.1** (15 points) Consider the following relation $R(A, B, C, D, E)$ with the following set of functional dependencies

$$\Sigma = \{\{C\} \rightarrow \{E\}, \{B\} \rightarrow \{E\}, \{B\} \rightarrow \{A\}, \{C, E\} \rightarrow \{D\},$$
$$\{B, E\} \rightarrow \{C\}, \{A, D\} \rightarrow \{B\}, \{A, D\} \rightarrow \{C\}\}$$

(a) (4 points) Write ALL the keys of $R$ with respect to $\Sigma$. (*choices in Examplify*)

**NOTES:**
The keys are

- $\{B\}$
- $\{A, C\}$
- $\{A, D\}$

(b) (3 points) Is $R$ in BCNF with respect to $\Sigma$? If not, write down all FDs in $\Sigma$ that violates the BCNF requirements. (*choices in Examplify*)

**NOTES:**
The violations are

- $\{C\} \rightarrow \{E\}$
- $\{C, E\} \rightarrow \{D\}$

(c) (5 points) Is $R$ in BCNF with respect to $\Sigma$? If not, apply the BCNF decomposition algorithm (*introduced in CS2102 lectures*) on $R$, and select the relations in the final result of your BCNF decomposition. (*choices in Examplify*)

**NOTES:**
The decomposed schema are

- $R1(A, B, C)$
- $R2(C, D, E)$

$R3(C, E)$ and $R3(D, E)$ are also possible decomposition when using the algorithm because we did not require subsumed relation to be removed from the result. No other relations are reachable using only the algorithm.

(d) (3 points) Is $R$ in 3NF with respect to $\Sigma$? If not, write down all FDs in $\Sigma$ that violates the 3NF requirements. (*choices in Examplify*)

**NOTES:**
The violations are

- $\{C\} \rightarrow \{E\}$

**4.2** (5 points) Consider the following relation $R(A, B, C, D, E, F)$ with the following set of functional dependencies

$$\Sigma = \{\{F\} \to \{D\}, \{B, F\} \to \{C\}, \{B, C, D, F\} \to \{E\}, \{D\} \to \{B\},\\ \{C, D\} \to \{A\}, \{B, D\} \to \{C\}, \{A, D\} \to \{C\}, \{B\} \to \{A\}\}$$

Derive a minimal basis of $\Sigma$. (*choices in Examplify*)

**NOTES:**

The minimal basis is the following set

- $\{B\} \to \{A\}$
- $\{D\} \to \{B\}$
- $\{D\} \to \{C\}$
- $\{F\} \to \{D\}$
- $\{F\} \to \{E\}$

These properties need to be satisfied:

- Must be a cover (*i.e., must be equivalent to the original set of FD*).
- The right-hand side must be *singular*.
- No redundant attributes.
- No redundant functional dependencies.

**4.3** (5 points) Consider an arbitrary relation $R(A, B, C, D, E, F)$. Suppose that $R$ is not in 3NF with respect to some unknown set of functional dependencies $\Sigma$. In that case, what is the maximum number of keys that $R$ can have? (*choices in Examplify*)

**NOTES:**

To derive this, note that one of the attributes cannot be part of a key. In that case, we can work with 5 attributes.

Note that if we include a set of attributes as a key, then all superset of this cannot be a key. As such, we want to keep the number of attributes to be the same in all keys. This gives us 5 choices for the size. To find the number of set with this size, we use binomial coefficient: ${}^{n}C_{k}$. We have 5 choices:

- ${}^{5}C_{1}$: 5
- ${}^{5}C_{2}$: 10
- ${}^{5}C_{3}$: 10
- ${}^{5}C_{4}$: 5
- ${}^{5}C_{5}$: 1

So the maximum is 10.