COMP 115
Databases
Fall 2019

Exam 1

October 21, 2019
4:30–5:45 PM

1. Schema Design (30 points)

We have the following badly designed schema for modeling university Departments, Majors, and Students. A Major is offered by exactly one Department. Each Student has exactly one Major.

```
create table university(
    dept_id int not null,
    dept_name varchar not null,
    dept_address varchar not null,
    major_id int not null,
    major_name varchar not null,
    major_dept_id int not null,
    student_id int not null primary key,
    student_name varchar not null,
    student_major_id int not null,
    student_graduation_date date not null
);
```

In your answers, please abbreviate column names, e.g. dept_id → DI, dept_name → DN, student_major_id → SMI, etc.

1.1. (3 points) What are the entities being modeled?

1.2. (3 points) What are the relationships among the entities? Identify the relationships as one-to-one, one-to-many, and many-to-many using arrow notation. E.g. `A ↔ B` is one-to-one, `A → B` and `A ← B` are one-to-many, and `A — B` is many-to-many.

1.3. (8 points) What functional dependencies exist? You should be able to identify these using:

- The information provided in the question.
- Your entity and relationship definitions.
- Naming hints (e.g. a column name ending in _id identifies something uniquely).

Guidelines:

- Do not list FDs that can be obtained by transitivity, e.g. if `A → B` and `B → C`, then don't bother listing `A → C`.
- Do not list trivial FDs, e.g. `A,B → A`.
- Remember that `A → B,C` is equivalent to `A → B` and `A → C`. I don't care which notation you use, but obviously `A → B,C` minimizes the amount of writing you have to do.

1.4. (16 points) Do a BCNF decomposition of the university table using the FDs you gave earlier. For each step:

- Identify a table that is not BCNF.
- Identify the FD that causes the BCNF violation.
- Identify the closure of that FD.
- Identify the two tables resulting from the decomposition.

2. SQL (30 points)

The following questions rely on this schema:

```
create table sale_date(
    when date not null primary key,
    weekday bool not null, -- True for a weekday
    weekend bool not null, -- True for a weekend day
    holiday bool not null  -- True for a holiday
);

create table salesperson(
    salesperson_id int not null primary key,
    name varchar not null
);

create table sale(
    sale_id int not null primary key,
    when date not null
        references sale_date,
    salesperson_id int not null
        references salesperson,
    price numeric(10, 2) not null
);
```

2.1. (5 points) Write a SQL query to print the names of salespeople who sold anything on 1/5/2018. Don't print any name more than once.

2.2. (5 points) Write a SQL query to print the average weekday sales, across all sales and dates, (i.e., `sale_date.weekday` is true).

2.3. (8 points) Write a SQL query to print the names of salespeople who have never made a sale.

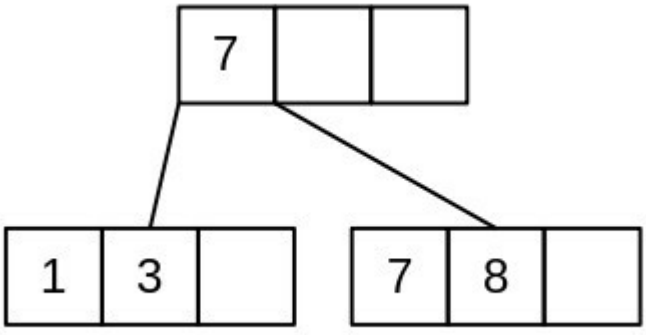
2.4. (12 points) Write a SQL query to print total sales for each sale date, ordered by date. Be sure to include days on which there were no sales.

3. B+-trees (24 points)

In this question, assume that a B+-tree leaf node can accommodate three records, and that an interior node can accommodate three keys. Highly unrealistic, but never mind that.

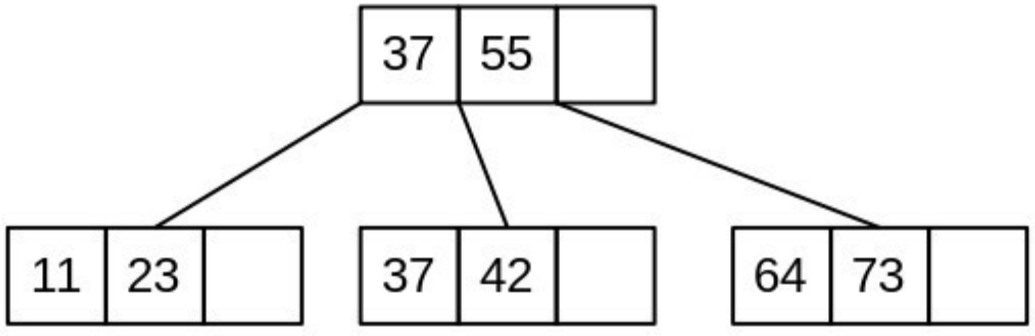
3.1. (4 points) Draw the B+-tree obtained by inserting records with keys 10 and 20 into an empty B+-tree.

3.2. (4 points) Given this B+-tree:



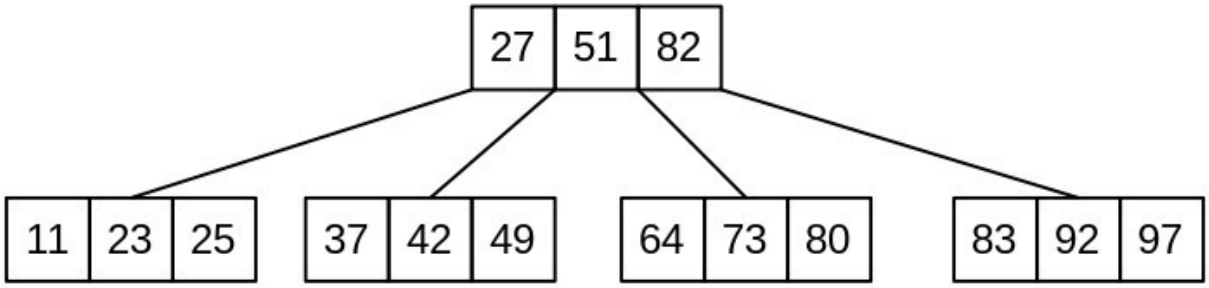
draw the B+-tree resulting from the insertion of records with keys 2 and 6. (Note: The B+-tree in the diagram has a root containing the key 7. The leaf nodes have the full record containing key 7, as well as records for all the other keys shown.)

3.3. (4 points) Given this B+-tree:



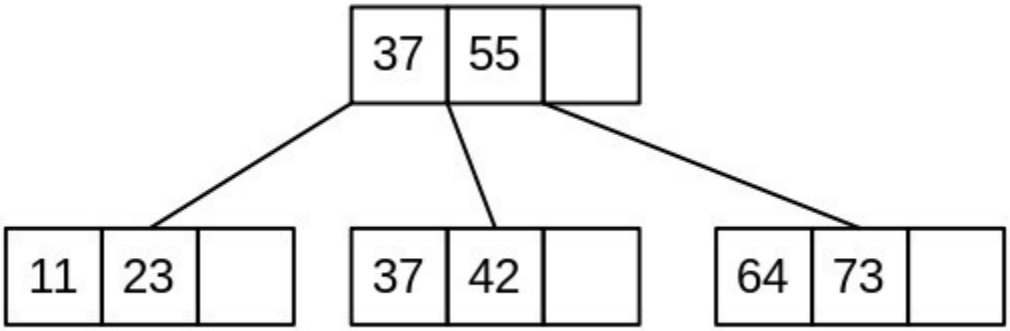
draw the B+-tree resulting from the insertion of records with keys 53 and 59.

3.4. (6 points) Given this B+-tree:



draw the B+-tree resulting from the insertion of a record with key 50.

3.5. (6 points) Given this B+-tree:



draw the B+-tree resulting from the deletion of the record with key 37.

4. Indexes (16 points)

We have the following table and indexes:

```
create table R(
    a int,
    b int,
    c int,
    d varchar
);

create index on R(a, b, c);

create index on R(c, a);
```

Both indexes are implemented using B+-trees, (which is the default in all database systems).

Which indexes can potentially be used with the following queries. You can refer to the indexes by their columns, e.g. `abc` indicates the index on `R(a, b, c)`, and `ca` describes the index on `R(c, a)`.

4.1. (2 points) `select * from R where a = 1 and b = 2 and c > 3`
4.2. (2 points) `select * from R where a = 1 and b = 2 and c = 3`
4.3. (2 points) `select * from R where a > 1 and b = 2 and c = 3`
4.4. (2 points) `select * from R where a > 1 and b > 2 and c = 3`
4.5. (2 points) `select * from R where a = 1 and b > 2`
4.6. (2 points) `select * from R where a > 1 and b = 2`
4.7. (2 points) `select * from R where a > 1 and c = 3`
4.8. (2 points) `select * from R where b = 2 and c = 3`