MET CS 669 Database Design and Implementation for Business
Lab 4 Template: Procedural SQL and Concurrency

Section One – Stored Procedures

1. *Create Table Structure* – Create the tables in the social networking schema, including all of their columns, datatypes, and constraints. Create sequences for each table; these will be used to generate the primary and foreign key values in Step #2.

```
222 CREATE TABLE Person (
223     person_id DECIMAL(12) PRIMARY KEY NOT NULL,
224     first_name VARCHAR(32) NOT NULL,
225     last_name VARCHAR(32) NOT NULL,
226     username VARCHAR(20) NOT NULL
227 );
228 CREATE TABLE Post (
229     post_id DECIMAL(12) PRIMARY KEY NOT NULL,
230     person_id DECIMAL(12) NOT NULL,
231     contents VARCHAR(255) NOT NULL,
232     created_on DATE NOT NULL,
233     summary VARCHAR(13) NOT NULL,
234     FOREIGN KEY (person_id) REFERENCES Person(person_id)
235 );
236 CREATE TABLE Likes (
237     likes_id DECIMAL(12) PRIMARY KEY NOT NULL,
238     person_id DECIMAL(12) NOT NULL,
239     post_id DECIMAL(12) NOT NULL,
240     liked_on DATE,
241     FOREIGN KEY (person_id) REFERENCES Person(person_id),
242     FOREIGN KEY (post_id) REFERENCES Post(post_id)
243 );
244 CREATE SEQUENCE person_seq START WITH 1;
245 CREATE SEQUENCE post_seq START WITH 1;
246 CREATE SEQUENCE likes_seq START WITH 1;
```

Data Output | Messages | Notifications

```
CREATE SEQUENCE

Query returned successfully in 233 msec.
```

2. *Populate Tables* – Populate the tables with data, ensuring that there are at least 5 people, at least 8 posts, and at least 4 likes. Make sure to use sequences to generate the primary and foreign key values. Most of the fields are self-explanatory. As far as the "content" field in Post, make them whatever you like, such as "Take a look at these new pics" or "Just arrived in the Bahamas", and set the summary as the first 10 characters of the content, followed by "…".

```
INSERT INTO Person(person_id, first_name, last_name, username)
VALUES(nextval('person_seq'), 'Bob', 'Jones', 'bjones');
INSERT INTO Post(post_id, person_id, contents, created_on, summary)
VALUES(nextval('post_seq'), currval('person_seq'), 'Look at my new motorcycle', CAST('01-MAY-2024' AS DATE), 'Look at my...');
INSERT INTO Post(post_id, person_id, contents, created_on, summary)
VALUES(nextval('post_seq'), currval('person_seq'), 'Just got back from grocery shopping', CAST('03-MAY-2024' AS DATE), 'Just got b...');
INSERT INTO Person(person_id, first_name, last_name, username)
VALUES(nextval('person_seq'), 'Susan', 'Smith', 'ssmith');
INSERT INTO Post(post_id, person_id, contents, created_on, summary)
VALUES(nextval('post_seq'), currval('person_seq'), 'My dog is so cute', CAST('02-MAY-2024' AS DATE), 'My dog is ...');
INSERT INTO Post(post_id, person_id, contents, created_on, summary)
VALUES(nextval('post_seq'), currval('person_seq'), 'On my way to a concert', CAST('07-MAY-2024' AS DATE), 'On my way ...');
INSERT INTO Person(person_id, first_name, last_name, username)
VALUES(nextval('person_seq'), 'Sarah', 'Connor', 'sconnor');
INSERT INTO Post(post_id, person_id, contents, created_on, summary)
VALUES(nextval('post_seq'), currval('person_seq'), 'It is so hot outside today', CAST('03-MAY-2024' AS DATE), 'It is so h...');
INSERT INTO Post(post_id, person_id, contents, created_on, summary)
VALUES(nextval('post_seq'), currval('person_seq'), 'Waiting for the weekend', CAST('06-MAY-2024' AS DATE), 'Waiting fo...');
INSERT INTO Person(person_id, first_name, last_name, username)
VALUES(nextval('person_seq'), 'Max', 'Williams', 'mwilliams');
INSERT INTO Post(post_id, person_id, contents, created_on, summary)
VALUES(nextval('post_seq'), currval('person_seq'), 'Going on a drive', CAST('04-MAY-2024' AS DATE), 'Going on a...');
INSERT INTO Person(person_id, first_name, last_name, username)
VALUES(nextval('person_seq'), 'Jessica', 'Wright', 'jwright');
INSERT INTO Post(post_id, person_id, contents, created_on, summary)
VALUES(nextval('post_seq'), currval('person_seq'), 'Bought new clothes today', CAST('07-MAY-2024' AS DATE), 'Bought new...');
-- likes
```

Data Output | Messages | Notifications

```
INSERT 0 1

Query returned successfully in 123 msec.
```

```
-- likes
INSERT INTO Likes(likes_id, person_id, post_id, liked_on)
VALUES(nextval('likes_seq'), (SELECT person_id FROM Person WHERE username='jwright'), (SELECT post_id FROM Post WHERE summary='Going on a...'), CAST(
INSERT INTO Likes(likes_id, person_id, post_id, liked_on)
VALUES(nextval('likes_seq'), (SELECT person_id FROM Person WHERE username='bjones'), (SELECT post_id FROM Post WHERE summary='Waiting fo...'), CAST(
INSERT INTO Likes(likes_id, person_id, post_id, liked_on)
VALUES(nextval('likes_seq'), (SELECT person_id FROM Person WHERE username='ssmith'), (SELECT post_id FROM Post WHERE summary='It is so h...'), CAST(
INSERT INTO Likes(likes_id, person_id, post_id, liked_on)
VALUES(nextval('likes_seq'), (SELECT person_id FROM Person WHERE username='ssmith'), (SELECT post_id FROM Post WHERE summary='Going on a...'), CAST(
```

Data Output | Messages | Notifications

```
INSERT 0 1

Query returned successfully in 123 msec.
```

3. *Create Hardcoded Procedure* – Create a stored procedure named “add_michelle_stella” which has no parameters and adds a person named “Michelle Stella” to the Person table. Execute the stored procedure, and list out the rows in the Person table to show that Michelle Stella has been added.

```
287 CREATE OR REPLACE PROCEDURE add_michelle_stella()
288 AS
289 $proc$
290     BEGIN
291         INSERT INTO Person(person_id, first_name, last_name, username)
292         VALUES(nextval('person_seq'), 'Michelle', 'Stella', 'mstella');
293     END;
294 $proc$ LANGUAGE plpgsql;
295 CALL add_michelle_stella();
296 SELECT * FROM Person;
```

Data Output | Messages | Notifications

	person_id [PK] numeric (12)	first_name character varying (32)	last_name character varying (32)	username character varying (20)
1	1	Bob	Jones	bjones
2	2	Susan	Smith	ssmith
3	3	Sarah	Connor	sconnor
4	4	Max	Williams	mwilliams
5	5	Jessica	Wright	jwright
6	6	Michelle	Stella	mstella

4. *Create Reusable Procedure* – Create a reusable stored procedure named “add_person” that uses parameters and allows you to insert any new person into the Person table. Execute the stored procedure with a person of your choosing, then list out the Person table to show that the person was added to the table.

```
297 CREATE OR REPLACE PROCEDURE add_person(
298     first_name_arg IN VARCHAR,
299     last_name_arg IN VARCHAR,
300     username_arg IN VARCHAR)
301     LANGUAGE plpgsql
302 AS
303 $reusableproc$
304 BEGIN
305     INSERT INTO Person(person_id, first_name, last_name, username)
306     VALUES(nextval('person_seq'), first_name_arg, last_name_arg, username_arg);
307 END;
308 $reusableproc$;
309 CALL add_person('Clark', 'Simpson', 'csimpson');
310
311 SELECT * FROM Person;
```

Data Output | Messages | Notifications

	person_id [PK] numeric (12)	first_name character varying (32)	last_name character varying (32)	username character varying (20)
1	1	Bob	Jones	bjones
2	2	Susan	Smith	ssmith
3	3	Sarah	Connor	sconnor
4	4	Max	Williams	mwilliams
5	5	Jessica	Wright	jwright
6	6	Michelle	Stella	mstella
7	7	Clark	Simpson	csimpson

5. *Create Deriving Procedure* – Create a reusable stored procedure named “add_post” that uses parameters and allows you to insert any new post into the Post table. Instead of passing in the summary as a parameter, derive the summary from the content, storing the derivation temporarily in a variable (which is then used as part of the insert statement). Recall that the summary field stores the first 10 characters of the content followed by “…”. Execute the stored procedure to add a post of your choosing, then list out the Post table to show that the addition succeeded.

```
311 CREATE OR REPLACE PROCEDURE add_post(
312     contents_arg IN VARCHAR,
313     created_on_arg IN DATE)
314     LANGUAGE plpgsql
315 AS
316 $$
317 DECLARE
318     v_summary VARCHAR(13);
319 BEGIN
320     v_summary := SUBSTRING(contents_arg FROM 1 FOR 10) || '...';
321     INSERT INTO Post(post_id, person_id, contents, created_on, summary)
322     VALUES(nextval('post_seq'), currval('person_seq'), contents_arg, created_on_arg, v_summary);
323 END;
324 $$;
325 CALL add_post('I love chocolate chip cookies', CAST('09-MAY-2024' AS DATE));
326 SELECT * FROM Post;
```

Data Output | Messages | Notifications

	post_id [PK] numeric (12)	person_id numeric (12)	contents character varying (255)	created_on date	summary character varying (13)
1	1	1	Look at my new motorcycle	2024-05-01	Look at my...
2	2	1	Just got back from grocery shopping	2024-05-03	Just got b...
3	3	2	My dog is so cute	2024-05-02	My dog is ...
4	4	2	On my way to a concert	2024-05-07	On my way ...
5	5	3	It is so hot outside today	2024-05-03	It is so h...
6	6	3	Waiting for the weekend	2024-05-06	Waiting fo...
7	7	4	Going on a drive	2024-05-04	Going on a...
8	8	5	Bought new clothes today	2024-05-07	Bought new...
9	9	7	I love chocolate chip cookies	2024-05-09	I love cho...

6. *Create Lookup Procedure* – Create a reusable stored procedure named “add_like” that uses parameters and allows you to insert any new “like”. Rather than passing in the person_id value as a parameter to identify which person is liking which post, pass in the

username of the person. The stored procedure should then lookup the person_id and store it in a variable to be used in the insert statement. Execute the procedure to add a “like” of your choosing, then list out the Like table to show the addition succeeded.

```
327 CREATE OR REPLACE PROCEDURE add_like(
328     username_arg IN VARCHAR,
329     post_id_arg IN DECIMAL,
330     liked_on_arg IN DATE)
331     LANGUAGE plpgsql
332 AS $$
333 DECLARE
334     v_person_id DECIMAL(12);
335 BEGIN
336     SELECT person_id
337     INTO v_person_id
338     FROM Person
339     WHERE username = username_arg;
340 
341     INSERT INTO Likes(likes_id, person_id, post_id, liked_on)
342     VALUES(nextval('likes_seq'), v_person_id, post_id_arg, liked_on_arg);
343 END;
344 $$;
345 CALL add_like('mstella', 4, CAST('08-MAY-2024' AS DATE));
346 SELECT * FROM Likes;
347
```

Data Output | Messages | Notifications

	likes_id [PK] numeric (12)	person_id numeric (12)	post_id numeric (12)	liked_on date
1	1	5	7	2024-05-07
2	2	1	6	2024-05-07
3	3	2	5	2024-05-04
4	4	2	7	2024-05-05
5	5	6	4	2024-05-08

Section Two – Triggers

7. *Single Table Validation Trigger* – One practical use of a trigger is validation within a single table (that is, the validation can be performed by using columns in the table being modified). Create a trigger that validates that the summary is being inserted correctly, that is, that the summary is actually the first 10 characters of the content followed by "…". The trigger should reject an insert that does not have a valid summary value. Verify the trigger works by issuing two insert commands – one with a correct summary, and one with an incorrect summary. List out the Post table after the inserts to show one insert was blocked and the other succeeded.

```
348 CREATE OR REPLACE FUNCTION invalid_summary_func()
349     RETURNS TRIGGER LANGUAGE plpgsql
350     AS $trigfunc$
351     BEGIN
352         RAISE EXCEPTION USING MESSAGE = 'Invalid summary value',
353         ERRCODE = 22000;
354     END;
355     $trigfunc$;
356 CREATE TRIGGER invalid_summary_trg
357 BEFORE UPDATE OR INSERT ON Post
358 FOR EACH ROW WHEN (NEW.summary != SUBSTRING(NEW.contents FROM 1 FOR 10) || '...')
359 EXECUTE PROCEDURE invalid_summary_func();
360
361 INSERT INTO Post(post_id, person_id, contents, created_on, summary)
362 VALUES(nextval('post_seq'), currval('person_seq'), 'Taking a trip to Mexico', CAST('05-MAY-2024' AS DATE), 'Taking a trip');
363
```

Data Output | Messages | Notifications

```
ERROR:  Invalid summary value
CONTEXT:  PL/pgSQL function invalid_summary_func() line 3 at RAISE

SQL state: 22000
```

```
348 CREATE OR REPLACE FUNCTION invalid_summary_func()
349     RETURNS TRIGGER LANGUAGE plpgsql
350     AS $trigfunc$
351     BEGIN
352         RAISE EXCEPTION USING MESSAGE = 'Invalid summary value',
353         ERRCODE = 22000;
354     END;
355     $trigfunc$;
356 CREATE TRIGGER invalid_summary_trg
357 BEFORE UPDATE OR INSERT ON Post
358 FOR EACH ROW WHEN (NEW.summary != SUBSTRING(NEW.contents FROM 1 FOR 10) || '...')
359 EXECUTE PROCEDURE invalid_summary_func();
360
361 INSERT INTO Post(post_id, person_id, contents, created_on, summary)
362 VALUES(nextval('post_seq'), currval('person_seq'), 'Taking a trip to Mexico', CAST('05-MAY-2024' AS DATE), 'Taking a t...');
363 SELECT * FROM Post;
364
```

Data Output | Messages | Notifications

	post_id [PK] numeric (12)	person_id numeric (12)	contents character varying (255)	created_on date	summary character varying (13)
1	1	1	Look at my new motorcycle	2024-05-01	Look at my...
2	2	1	Just got back from grocery shopping	2024-05-03	Just got b...
3	3	2	My dog is so cute	2024-05-02	My dog is ...
4	4	2	On my way to a concert	2024-05-07	On my way ...
5	5	3	It is so hot outside today	2024-05-03	It is so h...
6	6	3	Waiting for the weekend	2024-05-06	Waiting fo...
7	7	4	Going on a drive	2024-05-04	Going on a...
8	8	5	Bought new clothes today	2024-05-07	Bought new...
9	9	7	I love chocolate chip cookies	2024-05-09	I love cho...
10	10	7	Taking a trip to Mexico	2024-05-05	Taking a t...

8. *Cross-Table Validation Trigger* – Another practical use of a trigger is cross-table validation (that is, the validation needs columns from at least one table external to the table being updated). Create a trigger that blocks a “like” from being inserted if its “liked_on” date is before the post’s “created_on” date. Verify the trigger works by inserting two “likes” – one that passes this validation, and one that does not. List out the Likes table after the inserts to show one insert was blocked and the other succeeded.

```
364 CREATE OR REPLACE FUNCTION likes_date_func()
365     RETURNS TRIGGER LANGUAGE plpgsql
366     AS $$
367     DECLARE
368         v_created_on DATE;
369     BEGIN
370         SELECT created_on
371         INTO v_created_on
372         FROM Post
373         WHERE Post.post_id = NEW.post_id;
374 
375         IF NEW.liked_on < v_created_on THEN
376             RAISE EXCEPTION USING MESSAGE = 'Invalid liked on date',
377             ERRCODE = 22000;
378         END IF;
379         RETURN NEW;
380     END;
381     $$;
382 CREATE TRIGGER likes_date_trg
383 BEFORE UPDATE OR INSERT ON Likes
384 FOR EACH ROW
385 EXECUTE PROCEDURE likes_date_func();
386 
387 INSERT INTO Likes(likes_id, person_id, post_id, liked_on)
388 VALUES(nextval('likes_seq'), (SELECT person_id FROM Person WHERE username='mwilliams'),
389       (SELECT post_id FROM Post WHERE summary='It is so h...'), CAST('02-MAY-2024' AS DATE));
390 
```

Data Output | Messages | Notifications

```
ERROR:  Invalid liked on date
CONTEXT:  PL/pgSQL function likes_date_func() line 11 at RAISE

SQL state: 22000
```

```
364 CREATE OR REPLACE FUNCTION likes_date_func()
365     RETURNS TRIGGER LANGUAGE plpgsql
366     AS $$
367     DECLARE
368         v_created_on DATE;
369     BEGIN
370         SELECT created_on
371         INTO v_created_on
372         FROM Post
373         WHERE Post.post_id = NEW.post_id;
374 
375         IF NEW.liked_on < v_created_on THEN
376             RAISE EXCEPTION USING MESSAGE = 'Invalid liked on date',
377             ERRCODE = 22000;
378         END IF;
379         RETURN NEW;
380     END;
381     $$;
382 CREATE TRIGGER likes_date_trg
383 BEFORE UPDATE OR INSERT ON Likes
384 FOR EACH ROW
385 EXECUTE PROCEDURE likes_date_func();
386 
387 INSERT INTO Likes(likes_id, person_id, post_id, liked_on)
388 VALUES(nextval('likes_seq'), (SELECT person_id FROM Person WHERE username='mwilliams'),
389       (SELECT post_id FROM Post WHERE summary='It is so h...'), CAST('04-MAY-2024' AS DATE));
390 SELECT * FROM Likes;
391
```

Data Output | Messages | Notifications

	likes_id [PK] numeric (12)	person_id numeric (12)	post_id numeric (12)	liked_on date
2	2	1	6	2024-05-07
3	3	2	5	2024-05-04
4	4	2	7	2024-05-05
5	5	6	4	2024-05-08
6	6	4	5	2024-05-04

9. *History Trigger* – Another practical use of trigger is to maintain a history of values as they change. Create a table named post_content_history that is used to record updates to the content of a post, then create a trigger that keeps this table up-to-date

when updates happen to post contents. Verify the trigger works by updating a post's content, then listing out the post_content_history table (which should have a record of the update).

```
392 CREATE TABLE Post_content_history (
393     post_id DECIMAL(12) NOT NULL,
394     person_id DECIMAL(12) NOT NULL,
395     old_contents VARCHAR(255) NOT NULL,
396     old_summary VARCHAR(13) NOT NULL,
397     new_contents VARCHAR(255) NOT NULL,
398     new_summary VARCHAR(13) NOT NULL,
399     change_date DATE NOT NULL,
400     FOREIGN KEY (person_id) REFERENCES Person(person_id)
401 );
402
403 CREATE OR REPLACE FUNCTION post_history_func()
404 RETURNS TRIGGER LANGUAGE plpgsql
405 AS $$
406 BEGIN
407     IF OLD.contents <> NEW.contents THEN
408         INSERT INTO Post_content_history(post_id, person_id, old_contents, old_summary, new_contents, new_summary, change_date)
409         VALUES(NEW.post_id, NEW.person_id, OLD.contents, OLD.summary, NEW.contents, NEW.summary, CURRENT_DATE);
410     END IF;
411     RETURN NEW;
412 END;
413 $$;
414 CREATE TRIGGER post_history_trg
415 BEFORE UPDATE ON Post
416 FOR EACH ROW
417 EXECUTE PROCEDURE post_history_func();
418
419 UPDATE Post
420 SET contents = 'Bought some new clothes today', summary = 'Bought som...'
421 WHERE contents = 'Bought new clothes today';
422 SELECT * FROM Post_content_history;
423
```

Data Output | Messages | Notifications

	post_id numeric (12)	person_id numeric (12)	old_contents character varying (255)	old_summary character varying (13)	new_contents character varying (255)	new_summary character varying (13)	change_date date
1	8	5	Bought new clothes today	Bought new...	Bought some new clothes today	Bought som...	2024-06-06

Section Three – Normalization

10. *Creating Normalized Table Structure* – For this question, you create a set of normalized tables based upon the scenario given, and also identify some functional dependencies between the given fields….

a. Identify all functional dependencies in the set of fields listed above in the spreadsheet. These can be listed in the form of:

column1,column2,… □ column3, column4…

Make sure to explain your reasoning for the functional dependency choices.

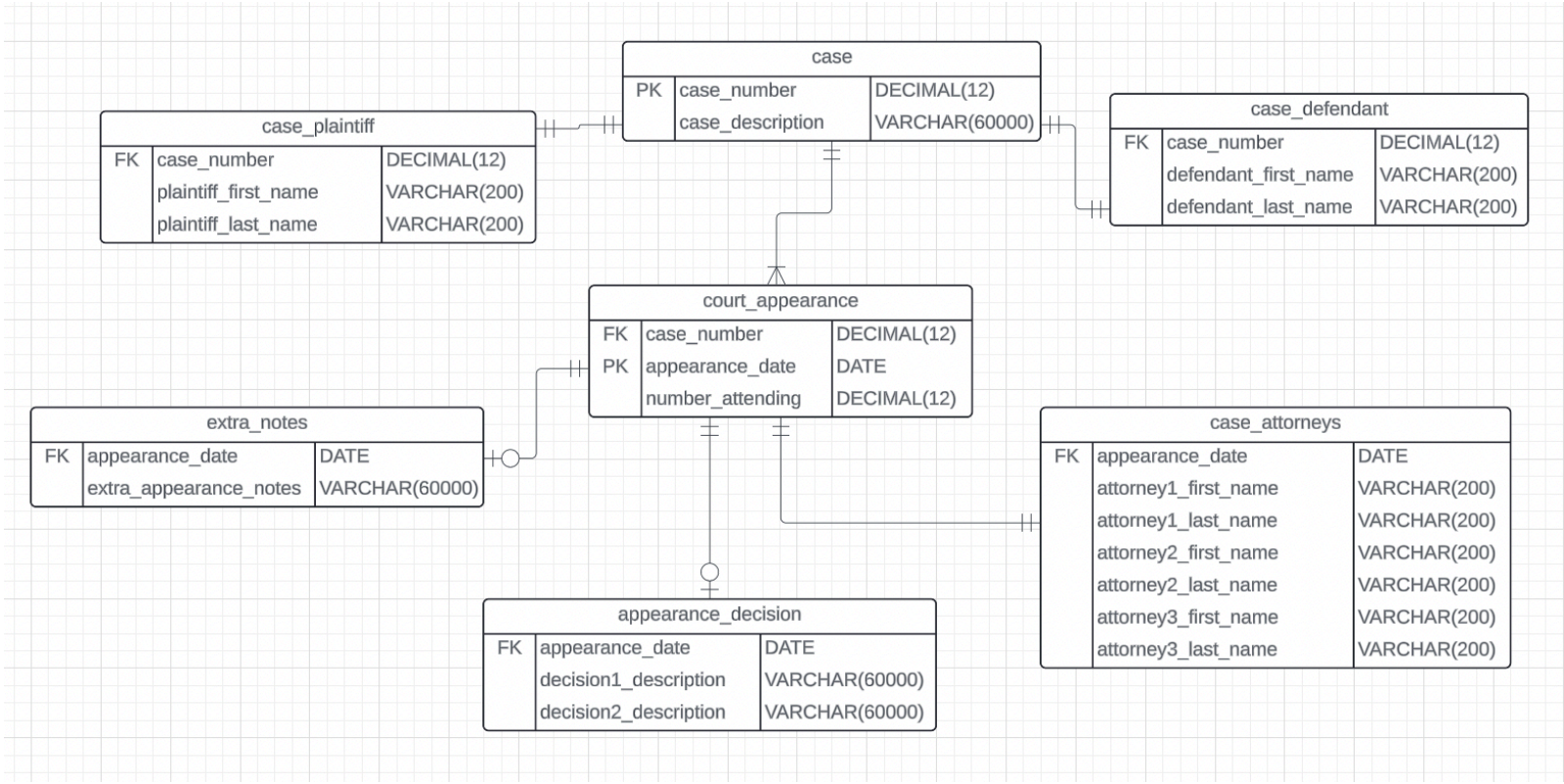
case_number, appearance date -> all attributes because this column combination is the candidate key as all the attributes can be identified by these two fields

case_number -> case_description, plaintiff_first_name, plaintiff_last_name, defendant_first_name, defendant_last_name because if we know the case number then we will know the case description and the plaintiffs and defendants

appearance_date -> number_attending, attorney1_first_name, attorney1_last_name, attorney2_first_name, attorney2_last_name, attorney3_first_name, attorney3_last_name, decision1_description, decision2_description, extra_appearance_notes because if we know the court appearance date then we know all of this information.

b. Suggest a set of normalized relational tables derived from how the court operates and the fields they store. Create a DBMS physical ERD representing this set of tables, which contains the entities, primary and foreign keys, attributes, relationships, and relationship constraints. You may add synthetic primary keys where needed. Make sure that the tables are normalized to BCNF, and to explain your choices.

Based on how the court operates and the fields they store, I would suggest tables that store the plaintiff's name by case number, the defendant's name by case number, and the case description by case number. More tables would include the court appearance date and attendance by case number, and then notes, attorneys, and decisions by court appearance date. These tables are BCNF normalized because the determinants and determined are in their own separate tables. Below is the physical DBMS ERD that I would create based on these tables.


case
PK
case_number
DECIMAL(12)
case_description
VARCHAR(60000)
case_plaintiff
FK
case_number
DECIMAL(12)
plaintiff_first_name
VARCHAR(200)
plaintiff_last_name
VARCHAR(200)
case_defendant
FK
case_number
DECIMAL(12)
defendant_first_name
VARCHAR(200)
defendant_last_name
VARCHAR(200)
court_appearance
FK
case_number
DECIMAL(12)
PK
appearance_date
DATE
number_attending
DECIMAL(12)
extra_notes
FK
appearance_date
DATE
extra_appearance_notes
VARCHAR(60000)
case_attorneys
FK
appearance_date
DATE
attorney1_first_name
VARCHAR(200)
attorney1_last_name
VARCHAR(200)
attorney2_first_name
VARCHAR(200)
attorney2_last_name
VARCHAR(200)
attorney3_first_name
VARCHAR(200)
attorney3_last_name
VARCHAR(200)
appearance_decision
FK
appearance_date
DATE
decision1_description
VARCHAR(60000)
decision2_description
VARCHAR(60000)