Question 3.3 (Putting it first for ease of checking):

```
-- Q3.3 Sequence identification (1 mark)
SELECT
        t.relname AS "Table Name",
        seq.relname AS "Sequence Name",
        ns.nspname AS "Schema"
FROM
        pg_class seq
        INNER JOIN pg_depend dep ON seq.oid = dep.objid
        INNER JOIN pg_class t ON dep.refobjid = t.oid
        INNER JOIN pg_namespace ns ON seq.relnamespace = ns.oid
WHERE
        seq.relkind = 'S'
        AND dep.deptype = 'a'
        AND t.relkind = 'r'
ORDER BY
        "Schema",
        "Table Name",
        "Sequence Name";
```

```
test=# -- Q3.3 Sequence identification (1 mark)
test=# SELECT
test-# t.relname AS "Table Name",
test-# seq.relname AS "Sequence Name",
test-# ns.nspname AS "Schema"
test-# FROM
test-# pg_class seq
test-# INNER JOIN pg_depend dep ON seq.oid = dep.objid
test-# INNER JOIN pg_class t ON dep.refobjid = t.oid
test-# INNER JOIN pg_namespace ns ON seq.relnamespace = ns.oid
test-# WHERE
test-# seq.relkind = 'S'
test-# AND dep.deptype = 'a'
test-# AND t.relkind = 'r'
test-# ORDER BY
test-# "Schema",
test-# "Table Name",
test-# "Sequence Name";
 Table Name |     Sequence Name     | Schema
------------+-----------------------+--------
 events     | events_event_id_seq   | public
 patrons    | patrons_patron_id_seq | public
(2 rows)
```

Question 1:

```
test=# -- Q1 CHECK Constraint (1 mark)
test=# ALTER TABLE Events
test-# ADD CONSTRAINT CK_EVENT_TYPE
test-# CHECK (event_type IN ('Loan', 'Return', 'Hold', 'Loss'));
ALTER TABLE
```

Question 2.1:

```
test=# CREATE OR REPLACE FUNCTION UDF_BI_GUARDIAN()
test-# RETURNS TRIGGER AS $guardian$
test$# BEGIN
test$# IF (NEW.dob > CURRENT_DATE - INTERVAL '18 years') THEN
test$#

test$# IF NEW.guardian IS NULL THEN
test$#

test$#

test$# RAISE EXCEPTION 'Patrons under 18 years of age require a guardian.';
test$#

test$# END IF;
test$#
test$#

test$# IF NOT EXISTS (SELECT 1 FROM Patrons WHERE patron_id = NEW.guardian AND dob
test$#

test$# <= CURRENT_DATE - INTERVAL '18 years') THEN
test$#

test$#

test$# RAISE EXCEPTION 'The specified guardian must be a registered patron and at least
test$#

test$#

test$# 18 years of age.';
test$#

test$# END IF;
test$# END IF;
test$# RETURN NEW;
test$# END;
test$# $guardian$ LANGUAGE plpgsql;
CREATE FUNCTION
test=#
test=# CREATE TRIGGER BI_GUARDIAN
test-# BEFORE INSERT ON Patrons
test-# FOR EACH ROW
test-# EXECUTE FUNCTION UDF_BI_GUARDIAN();
CREATE TRIGGER
```

Question 2.2:

```
test=# CREATE OR REPLACE FUNCTION UDF_BI_EMAIL_ADDR()
test-# RETURNS TRIGGER AS $email$
test$# BEGIN
test$# IF (NEW.dob <= CURRENT_DATE - INTERVAL '18 years') THEN
test$#

test$# IF NEW.email_address IS NULL OR TRIM(NEW.email_address = '') THEN
test$#

test$#

test$# RAISE EXCEPTION 'Patrons 18 years or older must provide an email address.';
test$#

test$# END IF;
test$# ELSE
test$#

test$# IF NEW.email_address IS NOT NULL THEN
test$#

test$#

test$# RAISE EXCEPTION 'Patrons under 18 years old must not provide an email address.';
test$#

test$# END IF;
test$# END IF;
test$# RETURN NEW;
test$# END;
test$# $email$ LANGUAGE plpgsql;
CREATE FUNCTION
test=#
test=# CREATE TRIGGER BI_EMAIL_ADDR
test-# BEFORE INSERT ON Patrons
test-# FOR EACH ROW
test-# EXECUTE FUNCTION UDF_BI_EMAIL_ADDR();
CREATE TRIGGER
```

Question 3.1:

```
test=# CREATE TRIGGER BI_EMAIL_ADDR
test-# BEFORE INSERT ON Patrons
test-# FOR EACH ROW
test-# EXECUTE FUNCTION UDF_BI_EMAIL_ADDR();
CREATE TRIGGER
test=# CREATE SEQUENCE ITEM_ID_SEQ
test-# START WITH 1000000000
test-# INCREMENT BY 1
test-# MINVALUE 1000000000
test-# MAXVALUE 9999999999
test-# NO CYCLE;
CREATE SEQUENCE
```

Question 3.2:

```
test=# CREATE OR REPLACE FUNCTION UDF_BI_ITEM_ID()
test-# RETURNS TRIGGER AS $item$
test$# DECLARE
test$# seq_num BIGINT;
test$# checksum INT;
test$# BEGIN
test$# seq_num := nextval('ITEM_ID_SEQ');
test$#
test$# checksum := 0;
test$# FOR i IN 1..10 LOOP
test$#

test$# checksum := checksum + substring(seq_num::text, i, 1)::int;
test$# END LOOP;
test$# checksum := checksum % 10;
test$#
test$# NEW.item_id := 'UQ' || seq_num::text || checksum::text;
test$# RETURN NEW;
test$# END;
test$# $item$ LANGUAGE plpgsql;
CREATE FUNCTION
test=#
test=# CREATE TRIGGER BI_ITEM_ID
test-# BEFORE INSERT ON Items
test-# FOR EACH ROW
test-# EXECUTE FUNCTION UDF_BI_ITEM_ID();
CREATE TRIGGER
```

Question 4.1:

```
test=# CREATE OR REPLACE FUNCTION UDF_BI_LOSS_CHARGE()
test-# RETURNS TRIGGER AS $loss$
test$# BEGIN
test$# IF NEW.event_type = 'Loss' THEN
test$#

test$# SELECT W.cost INTO NEW.charge
test$#

test$# FROM Works W
test$#

test$# JOIN Items I ON W.isbn = I.isbn
test$#

test$# WHERE I.item_id = NEW.item_id;
test$# END IF;
test$# RETURN NEW;
test$# END;
test$# $loss$ LANGUAGE plpgsql;
CREATE FUNCTION
test=#
test=# CREATE TRIGGER BI_LOSS_CHARGE
test-# BEFORE INSERT ON Events
test-# FOR EACH ROW
test-# EXECUTE FUNCTION UDF_BI_LOSS_CHARGE();
CREATE TRIGGER
```

Question 4.2:

```
test=# CREATE OR REPLACE FUNCTION UDF_AI_MISSING_RETURN()
test-# RETURNS TRIGGER AS $loan$
test$# DECLARE
test$# last_event RECORD;
test$# BEGIN
test$# SELECT * INTO last_event
test$# FROM Events
test$# WHERE item_id = NEW.item_id
test$# AND time_stamp < NEW.time_stamp
test$# ORDER BY time_stamp DESC
test$# LIMIT 1;
test$#
test$# IF last_event.event_type = 'Loan' THEN
test$#

test$# IF NEW.time_stamp <= last_event.time_stamp + INTERVAL '1 hour 1 millisecond' THEN
test$#

test$#

test$# RAISE EXCEPTION 'New loan cannot be within 1 hour and 1 millisecond of the last
test$#

test$#

test$# loan for the same item.';
test$#

test$# END IF;
test$#
test$#

test$# INSERT INTO Events (patron_id, item_id, event_type, time_stamp, charge)
test$#
OVERRIDING  SELECT      TABLE       VALUES
test$# VALUES (last_event.patron_id, last_event.item_id, 'Return', NEW.time_stamp - INTERVAL '1 hour', NULL);
test$# END IF;
test$# RETURN NEW;
test$# END;
test$# $loan$ LANGUAGE plpgsql;
CREATE FUNCTION
test=#
test=# CREATE TRIGGER AI_MISSING_RETURN
test-# AFTER INSERT ON Events
test-# FOR EACH ROW
test-# WHEN (NEW.event_type = 'Loan')
test-# EXECUTE FUNCTION UDF_AI_MISSING_RETURN();
CREATE TRIGGER
```

Question 4.3:

```
test=# CREATE OR REPLACE FUNCTION UDF_BI_HOLDS()
test-# RETURNS TRIGGER AS $$
test$# DECLARE
test$#     last_loan_time TIMESTAMP;
test$#     last_hold_time TIMESTAMP;
test$#     is_item_on_loan BOOLEAN;
test$#     can_item_be_held BOOLEAN;
test$#     item_is_lost BOOLEAN;
test$# BEGIN
test$#     SELECT EXISTS (
test$#         SELECT 1
test$#         FROM Events
test$#         WHERE item_id = NEW.item_id
test$#         AND event_type = 'Loss'
test$#         AND time_stamp <= NEW.time_stamp
test$#     ) INTO item_is_lost;
test$#
test$#     IF item_is_lost THEN
test$#         RAISE EXCEPTION 'A Hold cannot be placed on a lost item with ID %', NEW.item_id;
test$#     END IF;
test$#
test$#     SELECT time_stamp INTO last_loan_time
test$#     FROM Events
test$#     WHERE event_type = 'Loan' AND item_id = NEW.item_id
test$#     ORDER BY time_stamp DESC
test$#     LIMIT 1;
test$#
test$#     SELECT time_stamp INTO last_hold_time
test$#     FROM Events
test$#     WHERE event_type = 'Hold' AND item_id = NEW.item_id
test$#     ORDER BY time_stamp DESC
test$#     LIMIT 1;
test$#
test$#     is_item_on_loan := (
test$#         SELECT EXISTS (
test$#             SELECT 1
test$#             FROM Events
test$#             WHERE event_type = 'Loan' AND item_id = NEW.item_id
test$#             AND NOT EXISTS (
test$#                 SELECT 1 FROM Events
test$#                 WHERE event_type = 'Return' AND item_id = NEW.item_id
test$#                 AND time_stamp > last_loan_time
test$#             )
test$#         )
test$#     );
test$#
test$#     can_item_be_held := NOT is_item_on_loan;
test$#
test$#     IF NEW.event_type = 'Hold' THEN
test$#         IF last_hold_time IS NOT NULL AND (last_loan_time IS NULL OR last_hold_time > last_loan_time) THEN
test$#             RAISE EXCEPTION 'Consecutive holds are not permitted without a prior loan or return event.';
test$#         END IF;
test$#
test$#         IF NOT (can_item_be_held OR is_item_on_loan) THEN
test$#             RAISE EXCEPTION 'Cannot place hold: Item is neither available nor on loan.';
test$#         END IF;
test$#
test$#         IF is_item_on_loan THEN
test$#             NEW.time_stamp := last_loan_time + INTERVAL '42 days';
test$#         ELSE
test$#             NEW.time_stamp := NEW.time_stamp + INTERVAL '14 days';
test$#         END IF;
test$#     END IF;
test$#     RETURN NEW;
test$# END;
test$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
test=#
test=# CREATE TRIGGER BI_HOLDS
test-# BEFORE INSERT ON Events
test-# FOR EACH ROW
test-# WHEN (NEW.event_type = 'Hold')
test-# EXECUTE FUNCTION UDF_BI_HOLDS();
CREATE TRIGGER
```