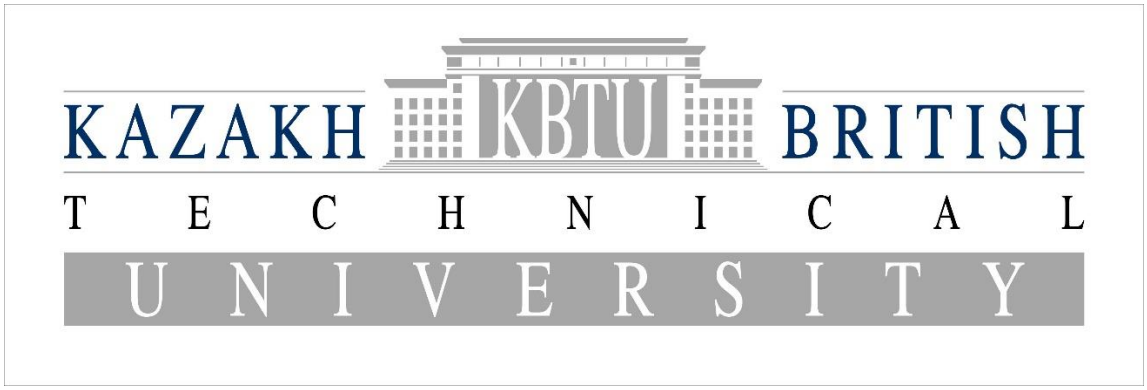

Kazakh-British Technical University
Faculty of Information Technology

Laboratory Work №7

Prepared by: Maratuly T.

Almaty, 2021

1) How can we store large-object types?

Answer:

Many database applications need to store attributes whose domain consists of large data items such as a photo, a high-resolution medical image, or a video. SQL, therefore, provides large-object data types for character data (clob) and binary data (blob). Usually use an SQL query to retrieve a “locator” for a large object and then use the locator to manipulate the object from the host language in which the application itself is written. For instance, the JDBC application program interface permits a locator to be fetched instead of the entire large object; the locator can then be used to fetch the large object in small pieces, rather than all at once, much like reading data from an operating system file using a read function call. When a query returns a large object, a pointer is returned rather than the large object itself.

2) What is the difference between privilege, role and user?

- Create accountant, administrator, support roles and grant appropriate privileges
- Create some users and assign them roles
- Give to some of them permission to grant roles to other users
- Revoke some privilege from particular user

Answer:

A role is a way to distinguish among various users as far as what these users can access/update in the database. Roles can own database objects and can assign privileges on those objects to other roles to control who has access to which objects. The SQL standard includes the privileges select, insert, update, and delete. We assign the roles to the users. The users are the people in database with corresponded roles along with the privileges

A)

```
-- A. create accountant, administrator, support roles and grant
appropriate privileges
-- Roles:
CREATE ROLE accountant; -- Бухгалтер ( Просмотреть аккаунты,
просмотреть транзакции, изменение баланса)
CREATE ROLE administrator; -- Администратор ( ВСЕ ПРАВА НА БАЗУ ДАННЫХ
)
CREATE ROLE support; -- Служба поддержки ( Просмотреть все, но ничего
не менять )
-- Privileges to Accountant ( SELECT - Бух Учет, Update - Закидывать
зарплату)
GRANT SELECT, UPDATE (balance) ON accounts TO accountant;
GRANT SELECT ON transactions TO accountant;
-- Privileges to Administrator ( All the privileges ON Database since
it is Administrator)
GRANT ALL PRIVILEGES ON DATABASE lab_seven TO administrator;
-- Privileges to Support ( Can see all the info but don't influence
them)
GRANT SELECT ON accounts,customers,transactions TO support;
```

B)
```
-- B Create some users and assign them roles,
CREATE USER zarina_kairatova;
CREATE USER temirbolat_maratuly;
CREATE USER alexander_luchin;

GRANT support to temirbolat_maratuly;
GRANT administrator to alexander_luchin;
GRANT accountant to zarina_kairatova;
```

C)
```
-- C Give to some of them permission to grant roles to other users

CREATE ROLE role_creator CREATEROLE;
GRANT role_creator to temirbolat_maratuly;
```

D)
```
-- D revoke some privilege from particular user

REVOKE SELECT ON customers FROM temirbolat_maratuly;
REVOKE UPDATE(balance) ON accounts FROM zarina_kairatova;
```

3) Add appropriate constraints

- check if transaction has same currency for source and destination accounts (use assertion)
- add not null constraints

Answer:

A)
```
-- A. check if transaction has same currency for source and
destination accounts (use assertion)

CREATE ASSERTION currency_assertion CHECK ( NOT EXISTS (
SELECT tr.src_account,tr.dst_account,ac.currency as
source_currency,ac_two.currency as destination_currency
FROM transactions tr
JOIN accounts ac
    ON tr.src_account = ac.account_id
JOIN accounts ac_two
    ON tr.dst_account = ac_two.account_id
WHERE ac.currency != ac_two.currency
) );
```

B)
```
-- B  add not null consraints
SELECT * FROM transactions;
-- First thought
ALTER TABLE transactions
ALTER COLUMN date SET NOT NULL;

ALTER TABLE transactions
ALTER COLUMN src_account SET NOT NULL;

ALTER TABLE transactions
ALTER COLUMN dst_account SET NOT NULL;

ALTER TABLE transactions
ALTER COLUMN amount SET NOT NULL;
```

```
ALTER TABLE transactions
ALTER COLUMN status SET NOT NULL;

-- Second thought
CREATE ASSERTION not_null_assertion CHECK ( NOT EXISTS (
SELECT *
FROM transactions
WHERE date IS NULL OR src_account IS NULL OR dst_account IS NULL OR
amount IS NULL OR status IS NULL;
) );
```

4. Change currency column type to user-defined in accounts table

```
-- 4 Change currency column type to user-defined in accounts table
CREATE TYPE Valuta AS (param CHAR(5));

ALTER TABLE accounts
ALTER COLUMN currency TYPE Valuta USING currency::valuta;
```

5. Create indexes:

- Index so that each customer can only have one account of one currency
- Index for searching transactions by currency and balance

Answer:

A.
```
-- A. index so that each customer can only have one account of one
currency
CREATE UNIQUE INDEX account_index ON accounts(customer_id,currency);
```

B.
```
-- B.  index for searching transactions by currency and balance
CREATE INDEX currency_balance_index ON accounts(currency,balance);
```

6. Write a SQL transaction that illustrates money transaction from one account to another:

- Create transaction with “init” status
- Increase balance for destination account and decrease for source account
- If in source account balance becomes below limit, then make rollback
- Update transaction with appropriate status(commit or rollback)

Answer:

```
-- 6  Write a SQL transaction that illustrates money transaction from one
account to another:
CREATE OR REPLACE PROCEDURE transfer_money(
    source_account varchar(40),
    destination_account varchar(40),
    money_number double precision
)
    LANGUAGE plpgsql
AS
    $$
DECLARE
    balance_sum double precision = (SELECT balance FROM accounts WHERE
account_id = source_account LIMIT 1);
DECLARE
    account_limit double precision = (SELECT "limit" FROM accounts WHERE
account_id = source_account LIMIT 1);
DECLARE
    new_transaction_id integer = (SELECT id + 1 FROM transactions ORDER BY id
DESC LIMIT 1 );
BEGIN
    -- create transaction with "init" status
    INSERT INTO transactions VALUES
(new_transaction_id,clock_timestamp(),source_account,destination_account,mone
y_number,'init');
   -- SAVEPOINT just_created_transaction_row;

    --  increase balance for destination account and decrease for source
account
    UPDATE accounts
    SET balance = balance + money_number
    WHERE account_id = destination_account;

    UPDATE accounts
    SET balance = balance - money_number
    WHERE account_id = source_account;

    --  if in source account balance becomes below limit, then make rollback
    IF balance_sum - money_number < account_limit THEN
        -- ROLLBACK TO SAVEPOINT just_created_transaction_row;
        ROLLBACK;
        INSERT INTO transactions VALUES
(new_transaction_id,clock_timestamp(),source_account,destination_account,mone
y_number,'rollback');
       -- UPDATE transactions
        --SET status = 'rollback'
        --WHERE id = new_transaction_id;
    ELSE
        UPDATE transactions
        SET status = 'commit'
        WHERE id = new_transaction_id;
    END IF;

   -- RELEASE SAVEPOINT just_created_transaction_row;
    COMMIT;
END;
    $$;

call transfer_money('AB10203','NT10204',50);
```