

**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.

1. 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

1. *-- 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;
2. *-- 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;
3. *-- C Give to some of them permission to grant roles to other users*CREATE ROLE role\_creator CREATEROLE;  
   GRANT role\_creator to temirbolat\_maratuly;
4. *-- 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:**

1. *-- 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  
   ) );
2. *-- 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:**

1. *-- A. index so that each customer can only have one account of one currency*CREATE UNIQUE INDEX account\_index ON accounts(*customer\_id*,*currency*);
2. *-- 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,money\_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,money\_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);