Федеральное государственное автономное образовательное учреждение высшего образования **«Национальный исследовательский университет ИТМО»**

Факультет Программной Инженерии и Компьютерной Техники

**КУРСОВАЯ РАБОТА**

**Информационные системы и базы данных**

3 этап

Преподаватель: Шешуков Дмитрий Михайлович

Выполнили: Леденцов Дмитрий Андреевич

Аталян Александр Эдуардович

Группа: P33081

Санкт-Петербург, 2023г

Реализовать даталогическую модель в реляционной СУБД PostgreSQL:

* Создать необходимые объекты базы данных
* Заполнить созданные таблицы тестовыми данными
* Сделать скрипты для:

создания/удаления объектов базы данных

заполнения/удаления созданных таблиц

* Обеспечить целостность данных при помощи средств языка DDL.
* Добавить в базу данных триггеры для обеспечения комплексных ограничений целостности
* Реализовать функции и процедуры на основе описания бизнес-процессов (из этапа №1)
* Произвести анализ использования созданной базы данных:

выявить наиболее часто используемые запросы к объектам базы данных

результаты представить в виде текстового описания

* Создать индексы и доказать, что они полезны для вашей базы данных:

доказательство должно быть приведено в виде текстового описания

Скрипты запросов

CREATE TABLE M\_USER

(

    ID    *serial* *PRIMARY KEY*,

    PASSWORD        *VARCHAR*(20) NOT NULL,

    LOGIN        *VARCHAR*(20) NOT NULL UNIQUE

);

CREATE INDEX ON M\_USER USING HASH(ID);

CREATE TABLE M\_CHAT

(

    ID    *serial* *PRIMARY KEY*,

    NAME *VARCHAR*(20) NOT NULL

);

CREATE INDEX ON M\_CHAT USING HASH(ID);

CREATE TABLE M\_MESSAGE

(

    ID    *serial* *PRIMARY KEY*,

    SENDER\_ID   *INTEGER* *REFERENCES* M\_USER,

    CHAT\_ID     *INTEGER* *REFERENCES* M\_CHAT *ON DELETE CASCADE* NOT NULL,

    DATA *TEXT* NOT NULL

);

CREATE INDEX ON M\_MESSAGE USING HASH(ID);

CREATE TABLE M\_USER\_IN\_CHAT

(

    USER\_ID   *INTEGER* *REFERENCES* M\_USER *ON DELETE CASCADE* NOT NULL,

    CHAT\_ID *INTEGER* *REFERENCES* M\_CHAT *ON DELETE CASCADE* NOT NULL,

    ROLE  *VARCHAR*(20) NOT NULL *DEFAULT* 'REGULAR'

);

CREATE INDEX ON M\_USER\_IN\_CHAT USING HASH(USER\_ID);

CREATE TABLE M\_USER\_STATUS

(

    USER\_ID   *INTEGER* *REFERENCES* M\_USER *ON DELETE CASCADE* NOT NULL,

    NAME  *VARCHAR*(20) NOT NULL UNIQUE,

    DESCRIPTION *VARCHAR*(20)

);

CREATE INDEX ON M\_USER\_STATUS USING HASH(USER\_ID);

CREATE TABLE M\_USER\_PROFILE

(

    USER\_ID   *INTEGER* unique *REFERENCES* M\_USER *ON DELETE CASCADE* NOT NULL,

    RATING *int* *default* 0

);

CREATE INDEX ON M\_USER\_PROFILE USING HASH(USER\_ID);

CREATE TABLE M\_USER\_ACTION

(

    USER\_ID   *INTEGER* *REFERENCES* M\_USER *ON DELETE CASCADE* NOT NULL,

    NAME *VARCHAR*(20) NOT NULL,

    DESCRIPTION *VARCHAR*(20),

    STATUS  *VARCHAR*(20) NOT NULL,

*TIME* *TIMESTAMP*

);

CREATE INDEX ON M\_USER\_ACTION USING HASH(USER\_ID);

CREATE TABLE M\_LOG

(

    DATA *VARCHAR*(20)

);

CREATE OR REPLACE FUNCTION register(\_login *varchar*(20), \_password *varchar*(20))

returns *int*

language plpgsql

as

$$

DECLARE \_id *int*;

BEGIN

INSERT INTO M\_USER (LOGIN,PASSWORD)

VALUES(\_login, \_password)

RETURNING ID INTO \_id;

RETURN \_id;

END;

$$;

CREATE OR REPLACE FUNCTION login(\_login *varchar*(20), \_password *varchar*(20))

returns *int*

language plpgsql

as

$$

DECLARE \_id *int* = NULL;

BEGIN

SELECT ID INTO \_id from M\_USER WHERE (LOGIN = \_login and PASSWORD = \_password);

IF (\_id IS NULL) THEN

    RAISE INFO 'No such user';

END IF;

INSERT INTO M\_USER\_STATUS(USER\_ID,NAME) VALUES(\_id,'ONLINE');

RETURN \_id;

END;

$$;

CREATE OR REPLACE PROCEDURE logout(\_login *varchar*(20), \_password *varchar*(20))

language plpgsql

as

$$

DECLARE \_id *int* = NULL;

BEGIN

SELECT ID INTO \_id from M\_USER WHERE (LOGIN = \_login and PASSWORD = \_password);

DELETE FROM M\_USER\_STATUS WHERE  (M\_USER\_STATUS.USER\_ID = \_id and M\_USER\_STATUS.NAME = 'ONLINE');

END;

$$;

CREATE OR REPLACE PROCEDURE vote\_on\_user(\_user\_id *int*, positive bool *default* true)

language plpgsql

as

$$

BEGIN

UPDATE M\_USER\_PROFILE SET RATING = CASE WHEN (positive) then RATING+1 else RATING-1 end

WHERE M\_USER\_PROFILE.USER\_ID = \_user\_id;

END;

$$;

CREATE OR REPLACE FUNCTION GET\_MESSAGES\_FOR\_USER\_IN\_CHAT(\_user\_id *int*, \_chat *int*)

returns TABLE(SENDER\_ID *int*, SENDER *VARCHAR*(20), MESSAGE *TEXT*)

language plpgsql

as

$$

BEGIN

   RETURN QUERY

   SELECT SENDER.ID,SENDER.LOGIN, M\_MESSAGE.DATA

    FROM M\_USER CUR\_USER

    JOIN M\_MESSAGE ON (M\_MESSAGE.CHAT\_ID = \_chat AND M\_MESSAGE.SENDER\_ID <> CUR\_USER.ID)

    JOIN M\_USER SENDER ON M\_MESSAGE.SENDER\_ID = SENDER.ID

    WHERE CUR\_USER.ID = \_user\_id;

END;

$$;

CREATE OR REPLACE FUNCTION get\_chats\_for\_user(\_user\_id *int*)

RETURNS SETOF M\_CHAT

language plpgsql

as

$$

BEGIN

   RETURN QUERY

   SELECT M\_CHAT.ID, M\_CHAT.NAME

   FROM M\_USER

    JOIN M\_USER\_IN\_CHAT ON M\_USER\_IN\_CHAT.USER\_ID = M\_USER.ID

    JOIN M\_CHAT ON M\_CHAT.ID = M\_USER\_IN\_CHAT.CHAT\_ID

    WHERE M\_USER.ID = \_user\_id;

END;

$$;

CREATE OR REPLACE FUNCTION register\_action()

RETURNS trigger AS

$$

BEGIN

   INSERT INTO M\_USER\_ACTION(USER\_ID, NAME, STATUS, *TIME*)

    VALUES(NEW.ID, 'user registered', 'OK', NOW()::*TIMESTAMP*);

    INSERT INTO M\_USER\_PROFILE(USER\_ID) VALUES(NEW.ID);

    RETURN NEW;

END;

$$ language plpgsql ;

CREATE OR REPLACE FUNCTION add\_message\_action()

RETURNS trigger AS

$$

BEGIN

   INSERT INTO M\_USER\_ACTION(USER\_ID, NAME, STATUS, *TIME*, DESCRIPTION)

    VALUES(NEW.SENDER\_ID, 'message sent', 'OK', NOW()::*TIMESTAMP*, 'message id: '||NEW.ID);

    RETURN NEW;

END;

$$ language plpgsql ;

CREATE OR REPLACE FUNCTION add\_chat\_action()

RETURNS trigger AS

$$

BEGIN

   INSERT INTO M\_USER\_ACTION(USER\_ID, NAME, STATUS, *TIME*, DESCRIPTION)

    VALUES(NEW.USER\_ID, 'chat create', 'OK', NOW()::*TIMESTAMP*, 'chat id: ' || NEW.CHAT\_ID);

    RETURN NEW;

END;

$$ language plpgsql ;

CREATE OR REPLACE FUNCTION join\_chat\_action()

RETURNS trigger AS

$$

BEGIN

   INSERT INTO M\_USER\_ACTION(USER\_ID, NAME, STATUS, *TIME*, DESCRIPTION)

    VALUES(NEW.USER\_ID, 'join chat', 'OK', NOW()::*TIMESTAMP*, 'chat id: ' || NEW.CHAT\_ID);

    RETURN NEW;

END;

$$ language plpgsql ;

CREATE OR REPLACE FUNCTION leave\_chat\_action()

RETURNS trigger AS

$$

BEGIN

   INSERT INTO M\_USER\_ACTION(USER\_ID, NAME, STATUS, *TIME*, DESCRIPTION)

   VALUES(OLD.USER\_ID, CASE WHEN (OLD.ROLE='CREATOR') then 'leave and delete' else 'leave chat' end, 'OK', NOW()::*TIMESTAMP*, 'chat id: ' || OLD.CHAT\_ID);

    RETURN OLD;

END;

$$ language plpgsql ;

CREATE TRIGGER UserRegistered AFTER INSERT ON M\_USER

FOR EACH ROW

EXECUTE PROCEDURE register\_action();

CREATE TRIGGER UserSendMessageInfo AFTER INSERT ON M\_MESSAGE

FOR EACH ROW EXECUTE PROCEDURE add\_message\_action();

CREATE TRIGGER ChatCreateInfo AFTER INSERT ON M\_USER\_IN\_CHAT

FOR EACH ROW

WHEN (NEW.ROLE = 'CREATOR')

EXECUTE PROCEDURE add\_chat\_action();

CREATE TRIGGER ChatJoinInfo AFTER INSERT ON M\_USER\_IN\_CHAT

FOR EACH ROW

EXECUTE PROCEDURE join\_chat\_action();

CREATE TRIGGER ChatLeaveInfo AFTER DELETE ON M\_USER\_IN\_CHAT

FOR EACH ROW

EXECUTE PROCEDURE leave\_chat\_action();

Триггеры

UserRegistered *– при регистрации, создание соответвующих записей в таблице профиль пользователя*

UserSendMessageInfo – *при отправке сообщения*

ChatCreateInfo – *при создании чата*

ChatJoinInfo – *при присоединении к чату*

*ChatLeaveInfo – при покидании чата*

*Все триггеры обновляют список действий пользователя*

Функции

register – регистрирует пользователя

login – проверяет логин пароль

get\_messages\_for\_user\_in\_chat – получить список всех сообщений для нужного пользователя в чате

get\_chats\_for\_user – получить список всех чатов которые видит пользователь

vote\_on\_user – поставить пользователю оценку

Анализ

Наиболее часто используемая информация хранится в таблицах M\_USER, M\_CHAT, M\_MESSAGE*,* так как почти каждый запрос содержит одну или

несколько обращений к ним. Например, оттуда берутся список чатов и сообщений . Для ускорения обращений к ним, были созданы индексы для ID

каждой из таблиц, так как большинство обращений происходит именно по

индексу. Поскольку диапазон значений нас не интересует, и поиск по конкретному ID, то оптимально использовать HASH индексирование.