# Курсовая работа по предмету "Базы данных"

Андрей Козлов, гр. 4538

25 января 2013г.

## 1 Постановка задачи

Некоторая компания предоставляет своим сотрудникам возможность бесплатно общаться по мобильному телефону. Для этого был заключен контракт с оператором мобильной связи на следующих условиях:

- компании выдается набор корпоративных номеров, общение между которыми не тарифицируется;
- цены на все остальные услуги устанавливаются в соответствии с действующими тарифами оператора.

Сотрудник компании может заключить договор на использование корпоративного номера. В договоре указывается тариф оператора, по которому будут тарифицироваться услуги. Один сотрудник может иметь более одного номера в любой момент времени.

Чтобы контролировать расходы компании на мобильную связь, требуется реализовать биллинговую систему, умеющую выполнять следующие операции:

- добавить новый корпоративный номер;
- добавить нового сотрудника;
- заключить контракт с сотрудником на использование корпоративного номера с указанным тарифным планом;
- расторгнуть контракт с сотрудником на использование корпоративный номер;
- сменить тарифный план контракта;
- расчитать расходы сотрудника за заданный период;
- расчитать все расходы сотрудника;
- вычислить процент расходов на личные разговоры.

Данная курсовая работа представляет собой реализацию такой биллинговой системы.

## 2 Модель

Список сущностей:

- сотрудник;
- номер корпоративного телефона;
- тарифный план оператора;
- услуга (звонок, sms/mms-сообщение, gprs-соединение и т.д.);
- единица тарификации услуги (секунды, штуки, кБ и т.д.);
- мобильная операция.

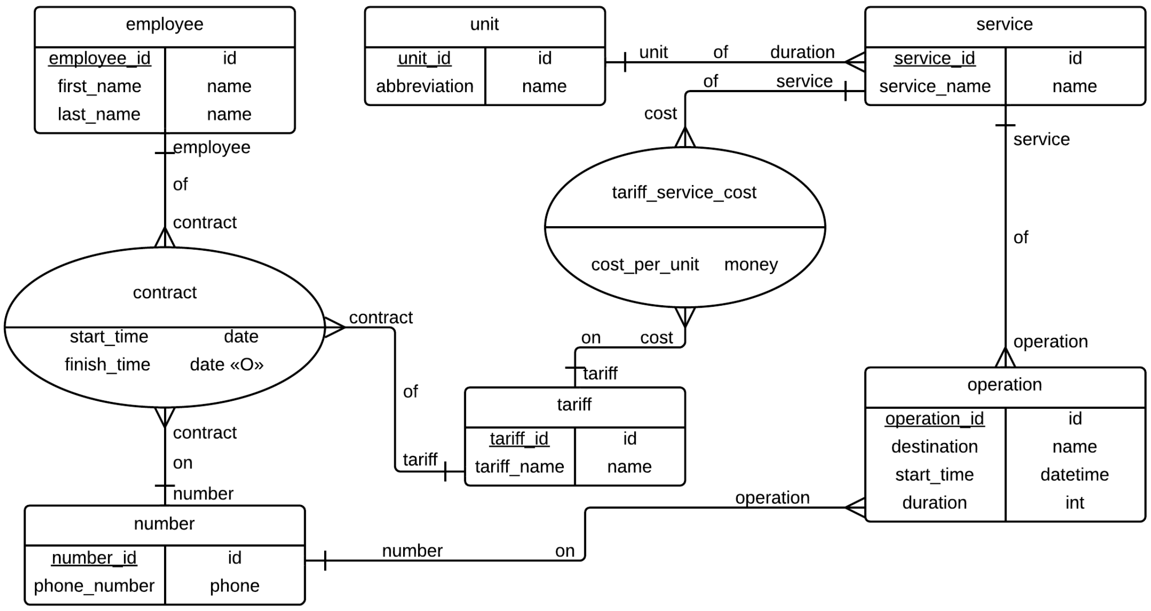

Рис. 1: Модель сущность-связь

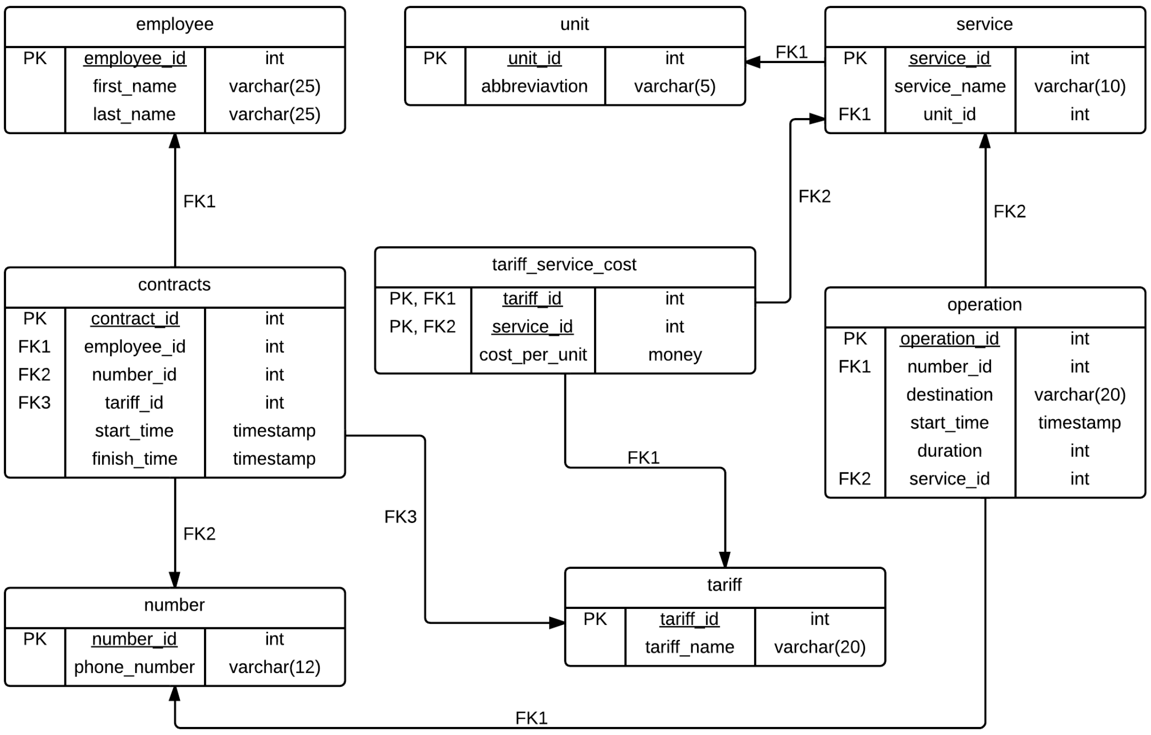

Рис. 2: Физическая модель

# 3 Исходный код

## 3.1 Создание таблиц

```

create table if not exists employees (
    employee_id serial primary key,
    first_name varchar(25) not null check (first_name != ''),
    last_name varchar(25) not null check (last_name != '')
);
create index lowercased_employees_idx on employees (lower(first_name),
    lower(last_name));

create table if not exists numbers (
    number_id serial primary key,
    phone_number varchar(12) not null unique check (phone_number != '')
);
create unique index numbers_idx on numbers (phone_number);

create table if not exists tariffs (
    tariff_id serial primary key,
    tariff_name varchar(20) not null unique check (tariff_name != '')
);

create function inf () returns timestamp as $$
    begin
        return '9999-12-31 23:59:59';
    end;
$$ language plpgsql;

create table if not exists contracts (
    employee_id int not null references employees (employee_id) on delete
        restrict on update restrict,
    number_id int not null references numbers (number_id) on delete
        restrict on update restrict,
    tariff_id int not null references tariffs (tariff_id) on delete
        restrict on update restrict,
    start_time timestamp not null,
    finish_time timestamp not null default inf() check (finish_time >=
        start_time)
);

create table if not exists units (
    unit_id serial primary key,
    abbreviation varchar(5) not null unique check (abbreviation != '')
);

create table if not exists services (
    service_id serial primary key,
    service_name varchar(10) not null unique check (service_name != '')
);

create table if not exists service_units (
    service_id int not null unique references services (service_id) on
        delete restrict on update restrict,
    unit_id int not null references units (unit_id) on delete restrict on
        update restrict
);
```

```

create table if not exists tariff_service_costs (
    tariff_id int not null references tariffs (tariff_id) on delete restrict on update restrict,
    service_id int not null references services (service_id) on delete restrict on update restrict,
    cost_per_unit money not null,
    unique (tariff_id, service_id)
);

create table if not exists operations (
    operation_id serial primary key,
    number_id int not null references numbers (number_id) on delete restrict on update restrict,
    destination varchar(20) not null,
    operation_time timestamp not null,
    duration int not null,
    service_id int not null references services (service_id) on delete restrict on update restrict
);
create index operations_idx on operations (number_id, operation_time, service_id);
```

## 3.2 Создание представлений, функций и триггеров

```

create or replace function get_employee_id (first_name_ varchar, last_name_ varchar) returns int as $$
    declare
        employee_id_ int;
    begin
        set transaction isolation level serializable read only;

        select employee_id into employee_id_ from employees where lower(first_name) = lower(first_name_) and lower(last_name) = lower(last_name_);

        if (employee_id_ is null) then
            raise exception 'there is no employee ''% %''', first_name_, last_name_;
        else
            return employee_id_;
        end if;
    end;
$$ language plpgsql;

create or replace function get_number_id (phone_number_ varchar) returns int as $$
    declare
        number_id_ int;
    begin
        set transaction isolation level serializable read only;

        select number_id into number_id_ from numbers where phone_number = phone_number_;

        if (number_id_ is null) then
```

```
            raise exception 'there is no phone number ''%''', phone_number_;
        else
            return number_id_;
        end if;
    end;
$$ language plpgsql;

create or replace function get_tariff_id (tariff_name_ varchar) returns int as $$
    declare
        tariff_id_ int;
    begin
        set transaction isolation level serializable read only;

        select tariff_id into tariff_id_ from tariffs where lower(tariff_name) = lower(tariff_name_);

        if (tariff_id_ is null) then
            raise exception 'there is no tariff ''%''', tariff_name_;
        else
            return tariff_id_;
        end if;
    end;
$$ language plpgsql;

create or replace function new_employee (first_name_ varchar, last_name_ varchar) returns int as $$
    begin
        set transaction isolation level serializable read write;

        insert into employees (first_name, last_name) values (first_name_, last_name_);
        return currval('employees_employee_id_seq');
    end;
$$ language plpgsql;

create or replace function new_number (phone_number_ varchar) returns int as $$
    begin
        set transaction isolation level serializable read write;

        insert into numbers (phone_number) values (phone_number_);
        return currval('numbers_number_id_seq');
    end;
$$ language plpgsql;

create or replace function open_contract_on_date (employee_id_ int, number_id_ int, tariff_id_ int, start_time_ timestamp) returns void as $$
    begin
        set transaction isolation level serializable read write;

        insert into contracts (employee_id, number_id, tariff_id, start_time) values (employee_id_, number_id_, tariff_id_, start_time_);
    end;
$$ language plpgsql;
```

```

create or replace function open_contract_now (employee_id_ int, number_id_ int, tariff_id_ int) returns void as $$
    begin
        set transaction isolation level serializable read write;

        insert into contracts (employee_id, number_id, tariff_id, start_time) values (employee_id_, number_id_, tariff_id_, now());
    end;
$$ language plpgsql;

create or replace function check_contract_is_open () returns trigger as $$
    begin
        if exists (select * from contracts where employee_id = new.employee_id and number_id = new.number_id and finish_time = inf()) then
            raise exception 'contract of employee ''%'' on number ''%'' is already open', new.employee_id, new.number_id;
        end if;
        return new;
    end;
$$ language plpgsql;

create trigger check_contract_is_open before insert on contracts
    for each row execute procedure check_contract_is_open();

create or replace function close_contract_on_date (employee_id_ int, number_id_ int, finish_time_ timestamp) returns void as $$
    begin
        set transaction isolation level serializable read write;

        update contracts set finish_time = finish_time_ where employee_id = employee_id_ and number_id = number_id_ and finish_time = inf();
    end;
$$ language plpgsql;

create or replace function close_contract_now (employee_id_ int, number_id_ int) returns void as $$
    begin
        set transaction isolation level serializable read write;

        perform close_contract_on_date (employee_id_, number_id_, now());
    end;
$$ language plpgsql;

create or replace function check_contract_is_closed () returns trigger as $$
    begin
        if not exists (select * from contracts where employee_id = new.employee_id and number_id = new.number_id and finish_time = inf()) then
            raise exception 'there is no opened contract of employee ''%'' on number ''%''', new.employee_id, new.number_id;
        end if;
        return new;
    end;
```

```
$$ language plpgsql;

create trigger check_contract_is_closed before update on contracts
    for each row execute procedure check_contract_is_closed ();

create or replace function change_tariff (number_id_ int, new_tariff_id int
    ) returns int as $$
    declare
        contract_id_ int;
        employee_id_ int;
        old_tariff_id int;
    begin
        set transaction isolation level serializable read write;

        select contract_id into contract_id_ from contracts where number_id
            = number_id_ and finish_time = inf();

        if (contract_id_ is null) then
            raise exception 'there is no opened contract on number ''%''',
                number_id_;
        else
            select tariff_id into old_tariff_id from contracts where
                contract_id = contract_id_;

            if (old_tariff_id != new_tariff_id) then
                update contracts set finish_time = current_timestamp where
                    contract_id = contract_id_;
                select employee_id into employee_id_ from contracts where
                    contract_id = contract_id_;
                return open_contract_now (employee_id_, number_id_,
                    new_tariff_id);
            else
                raise notice 'tariff on number ''%'' is already ''%''',
                    number_id_, old_tariff_id;
                return contract_id_;
            end if;
        end if;
    end;
$$ language plpgsql;

create or replace view all_contracts as
    select * from contracts natural join employees natural join numbers
        natural join tariffs;

create or replace function all_employee_contracts (first_name_ varchar,
    last_name_ varchar) returns table(phone_number varchar, tariff_name
    varchar, start_time timestamp, finish_time timestamp) as $$
    begin
        set transaction isolation level serializable read only;

        return query select c.phone_number, c.tariff_name, c.start_time, c.
            finish_time from all_contracts as c where employee_id =
            get_employee_id (first_name_, last_name_);
    end;
$$ language plpgsql;

create or replace function open_employee_contracts (first_name_ varchar,
    last_name_ varchar) returns table(phone_number varchar, tariff_name
    varchar, start_time timestamp) as $$
    begin
        set transaction isolation level serializable read only;

        return query select c.phone_number, c.tariff_name, c.start_time
            from all_employee_contracts(first_name_, last_name_) as c where
             finish_time = inf();
    end;
$$ language plpgsql;

create or replace view all_operations as
    select * from operations natural join contracts natural join employees
        natural join tariff_service_costs;

create or replace function total_employee_charges(first_name_ varchar,
    last_name_ varchar) returns money as $$
    declare
        charges money;
    begin
        set transaction isolation level serializable read only;

        select sum(duration * cost_per_unit) into charges from
            all_operations where employee_id = get_employee_id(first_name_,
             last_name_) group by employee_id;
        return coalesce(charges, 0.00::money);
    end;
$$ language plpgsql;

create or replace function period_employee_charges(first_name_ varchar,
    last_name_ varchar, start_time_ timestamp, finish_time_ timestamp)
    returns money as $$
    declare
        charges money;
    begin
        set transaction isolation level serializable read only;

        select sum(duration * cost_per_unit) into charges from
            all_operations where employee_id = get_employee_id(first_name_,
             last_name_) and operation_time >= start_time_ and
            operation_time < finish_time group by employee_id;
        return coalesce(charges, 0.00::money);
    end;
$$ language plpgsql;

create or replace function employee_charges_percent(first_name_ varchar,
    last_name_ varchar) returns real as $$
    declare
        charges money;
        total_charges money;
    begin
        set transaction isolation level serializable read only;

        select sum(duration * cost_per_unit) into charges from
            all_operations where employee_id = get_employee_id(first_name_,
             last_name_) and destination not in (select phone_number from
            numbers) group by first_name, last_name;

```

```sql
        select total_employee_charges (first_name_, last_name_) into total_charges;

        if (total_charges = 0.00::money) then
            raise notice 'employee ''% %'' has no operations yet', first_name_, last_name_;
            return 0.0;
        else
            return charges / total_charges;
        end if;
    end;
$$ language plpgsql;

create or replace function spenders () returns table (first_name varchar, last_name varchar, percent real) as $$
    begin
        set transaction isolation level serializable read only;

        return query select e.first_name, e.last_name, employee_charges_percent (e.first_name, e.last_name) as p from employees as e order by p desc;
    end;
$$ language plpgsql;

create or replace view all_operations_with_services as
    select * from all_operations natural join services natural join service_units natural join units natural join numbers;

create or replace function period_employee_operations (first_name_ varchar, last_name_ varchar, start_time_ timestamp, finish_time_ timestamp) returns table (phone_number varchar, destination varchar, operation_time timestamp, cost money, duration int, unit varchar, service_name varchar) as $$
    begin
        set transaction isolation level serializable read only;

        return query select o.phone_number, o.destination, o.operation_time, o.duration * o.cost_per_unit, o.duration, o.abbreviation, o.service_name from all_operations_with_services as o where employee_id = get_employee_id (first_name_, last_name_) and o.operation_time >= start_time_ and o.operation_time < finish_time_;
    end;
$$ language plpgsql;

create or replace function period_employee_work_operations (first_name_ varchar, last_name_ varchar, start_time_ timestamp, finish_time_ timestamp) returns table (phone_number varchar, destination varchar, operation_time timestamp, cost money, duration int, unit varchar, service_name varchar) as $$
    begin
        set transaction isolation level serializable read only;

        return query select * from period_employee_operations (first_name_, last_name_, start_time_, finish_time_) as o where o.destination in (select n.phone_number from numbers as n);
    end;
$$ language plpgsql;
```

```

create or replace function period_employee_not_work_operations (first_name_
     varchar, last_name_ varchar, start_time_ timestamp, finish_time_
    timestamp) returns table (phone_number varchar, destination varchar,
    operation_time timestamp, cost money, duration int, unit varchar,
    service_name varchar) as $$
    begin
        set transaction isolation level serializable read only;

        return query select * from period_employee_operations (first_name_,
            last_name_, start_time_, finish_time_) except all (select *
            from period_employee_work_operations (first_name_, last_name_,
            start_time_, finish_time_));
    end;
$$ language plpgsql;
```