Попов Владимир Вячеславович

# 1. Теоретический вопрос

## Транзакции. Параллельное исполнение. Уровни изоляции

## Ответ

Транзакции могут выполняться параллельно, при этом каждая транзакция должна исполняться как будто она одна в системе и транзакции корректные по отдельности должны быть корректны в совокупности. Для обеспечения этих условий используются: блокировки и синхронизация.

Какие проблемы могут возникнуть?  
1) Потерянное обновление  
2) Незафиксированные изменения

3) Потерянное обновление  
4) Несогласованное состояние

Блокировки:

В базах данных традиционно делают двухуровневые блокировки:

1. отсутствие блокировки
2. разделяемая на чтение
3. эксклюзивная на чтение и запись

Что делать во время дедлока?  
 1) Откатываем транзакцию(например с помощью таймаута)

2) Предотвращение взаимной блокировки.

Упорядочиваемость:

1. Любая последовательность исполнений эквивалентна какого-нибудь последовательному исполнению.

Cтрогий протокол двухфазной блокировки гарантирует упорядочиваемость.   
Что такое двухфазная блокировка?  
 1) Получаем все блокировки до операции

1. Если отпустили блокировку, то не возьмем

Типы блокировок:

1. блокировки на запись
2. блокировки таблицы
3. блокировка базы данных(при изменении схемы бд)

Из-за очевидного желания бОльшей производительности мы хотим допустить параллельное исполнение операций. Транзакции позволяют сделать это исполнение безопасным

Правила транзакций:

* Атомарность - транзакция или выполнена или нет, частичное выполнение невозможно.
* Согласованность - после каждой транзакции БД обязательно должна быть в согласованном состоянии. (если нет, транзакция очевидно откатывается)
* Изоляция - транзакции друг о друге не знают и не взаимодействуют.
* Устойчивость - гарантия, что принятые изменения (от зафиксированной транзакции) не могут быть потеряны.
* Транзакция должна думать, что исполняется одна
* Корректные по отдельности транзакции должны быть корректны в совокупности

Типы конфликтов

* Потерянное обновление(запись-запись) - пишем два раза, один из них перезаписывает другой
* Незафиксированное изменение(запись-чтение) - читаем значения, которые потом возможно будут откачены
* Несогласованное состояние(чтение-запись) - пишем под условием с двух потоков, условие может выполняться в каждом, но два раза апдейтить будет некорректно

Обычно изоляция транзакций происходит за счет версионирования / блокировок, в зависимости от идеологии БД.

Но если мы будем сильно изолировать транзакции, потеряем перфоманс. Идея: будем подбирать уровень изоляции под транзакцию самостоятельно.

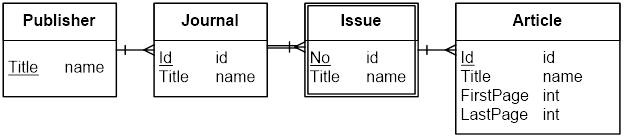
Уровни изоляции транзакций по убывающей гарантий и возрастающей перфоманса:

* Serializable - гарантия на все, аналогия взять блокировку на чтение и запись на все используемые ресурсы
* Snapshot (только в версионниках) - изоляция на уровне таблицы, фантомной записи уже не будет, но допускает аномалию “косая запись” - под условием пишем одновременно в два места и ломаемся
* Repeatable Read - гарантия что прочитаем из одного места два раза одно и то же значение. Новые записи в таблице видны не будут. Аномалия “фантомная запись”
* Read commited - гарантия что прочитаем закоммиченные какой-то транзакцией значения. То есть уже не мусор, но можно два раза прочитать разные значения. Аномалия “неповторяемое чтение”
* Read uncommitted - читаем то что сейчас есть в бд, даже неактуальные значения поставленные левой транзакцией, которая потом будет откатана. Аномалия “грязное чтение”

# 2. Теоретическое задание

## **Построение физической модели по концептуальной**

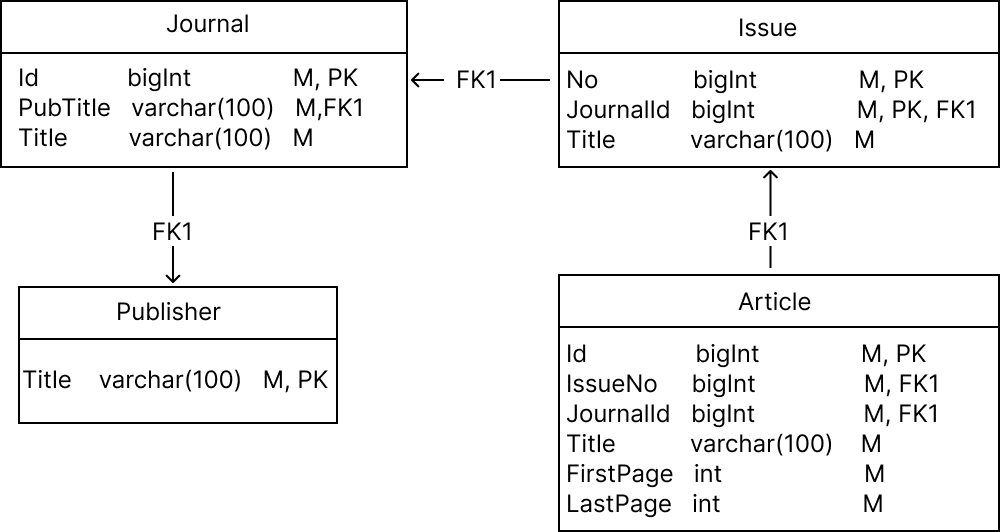
Постройте физическую модель данных по приведенной ER-диаграмме.



## Решение

Ваше решение

(от руки получилось некрасиво, к сожалению, сделал в Figma)



# 3. Практическое задание

Реализация хранимых процедур и функций  
Реализуйте хранимую процедуру, которая переводит человека из аутстаффа в штат с сохранением истории.

## Решение

create or replace procedure *migrateOutstaffWorkerToStaff*(

outstaffId bigint,

newStaffId bigint

)

language plpgsql

as $$

begin

insert into staffworker(

staffworkerid,

staffworkername,

staffworkergender,

staffworkerbirthdate,

staffworkerpassport,

staffworkerpositionid,

staffworkerphoto,

staffworkerphone,

staffworkercreatedat,

staffworkerfiredat,

staffworkerwarehouseid,

staffworkerdeactivationreason

) select

newStaffId,

outstaffworkername,

outstaffworkergender,

outstaffworkerbirthdate,

outstaffworkerpassport,

outstaffworkerpositionid,

outstaffworkerphoto,

outstaffworkerphone,

*now*(),

outstaffworkerfiredat,

outstaffworkerwarehouseid,

CASE

WHEN outstaffworkerdeactivationreason = 'incident' THEN 'incident'

WHEN outstaffworkerdeactivationreason = 'fired\_by\_comp' THEN 'incident'

WHEN outstaffworkerdeactivationreason = 'end\_of\_contract' THEN 'by\_agreement'

ELSE NULL END

from outstaffworker o

where o.outstaffworkerid = outstaffId;

update logentity

set logentitystaffworkerid = newStaffId,

logentityoutstaffworkerid = null

where logentityoutstaffworkerid = outstaffId;

delete from outstaffworker

where outstaffworkerid = outstaffId;

end;$$;