БАЗЫ ДАННЫХ

Лекция 6

ПЛАН НА СЕГОДНЯ

- Partishioned hash
- Индекс индексов
- KD-деревья
- Quad-деревья
- Введение в транзакции

PARTITIONED HASH

- Как одномерный хеш-индекс
- Варим хеш из нескольких компонетов
- Поровну или пропорционально

КОГДА ПОЛЕЗЕН

- Когда надо точно искать по паре полей
- Два отдельных хеш-индекса "скомбинировать" не выйдет
- Опираясь на один - можем получить высокую дублируемость
- А парный может быть - самое то

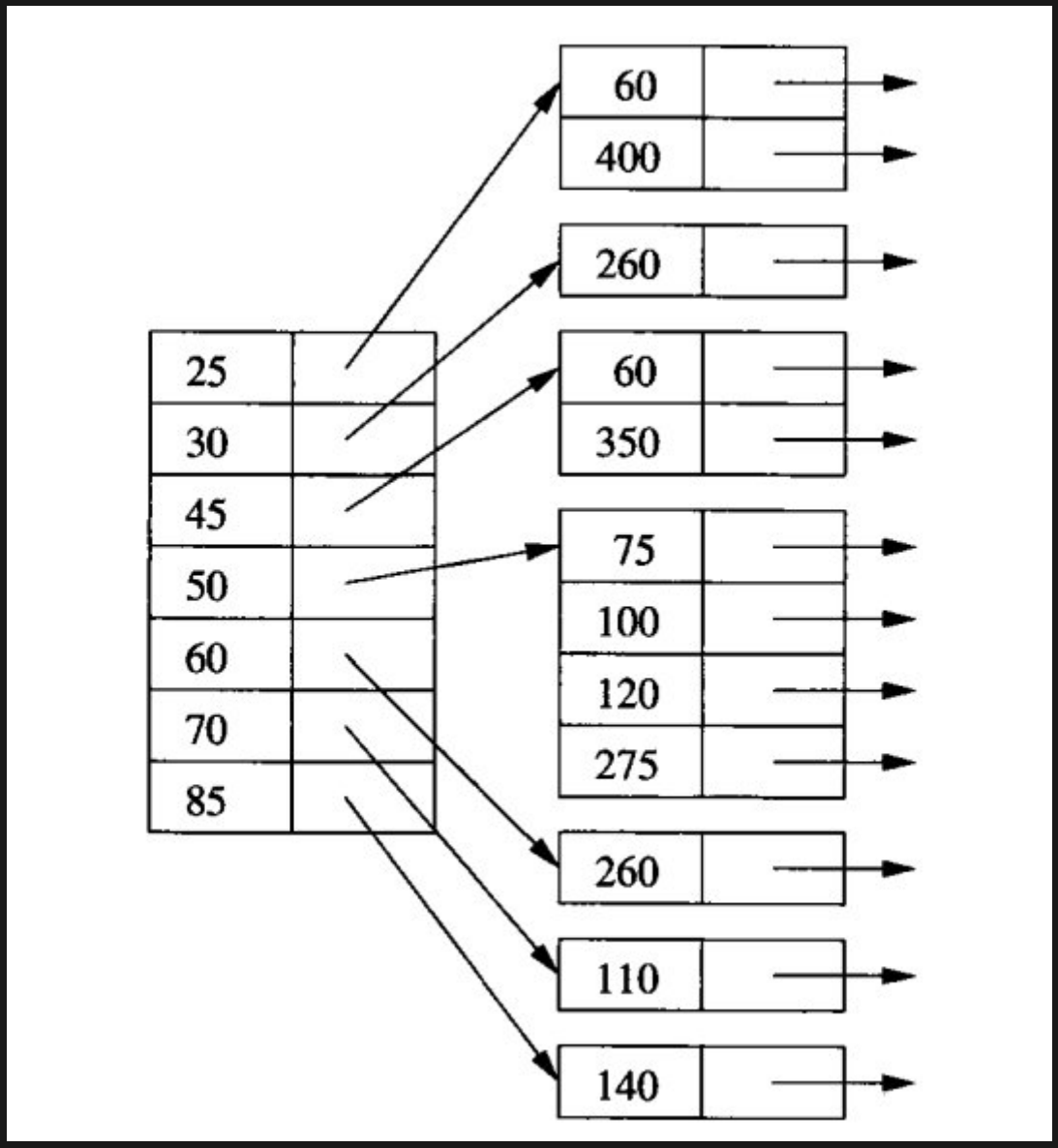
НЮАНСЫ

- Биты надо открывать по очереди
- Возможно, пропорционально доле
- Можно слить вместе не только две компоненты

ДЕРЕВО ДЕРЕВЬЕВ

- Построим B-дерево по одному атрибуту
- В листьях будем хранить B-деревья по второму атрибуту
- Надо, чтобы в деревьях "второго порядка" было достаточно много элементов
- Ради этого можно "приподнять" первое дерево

КАРТИНКА


25
30
45
50
60
70
85
60
400
260
60
350
75
100
120
275
260
110
140

ДЕРЕВО ДЕРЕВЬЕВ

- На таком индексе можно сочетать парные и одиночные запросы
- Если одиночный запрос делается по первому индексу
- В этом смысле не все равно, кто будет первым

ДЕРЕВО ДЕРЕВЬЕВ

- Вторым индексом может быть хеш-индекс
- Удобно для range-запросов по первому атрибуту и точечному по второму
- Или одномерный грид

KD-ДЕРЕВЬЯ

- Начнем с in-memory
- Есть csv-файл, влезающий в память
- Но очень большой
- Хотим организовать двумерный индекс по количественным столбцам

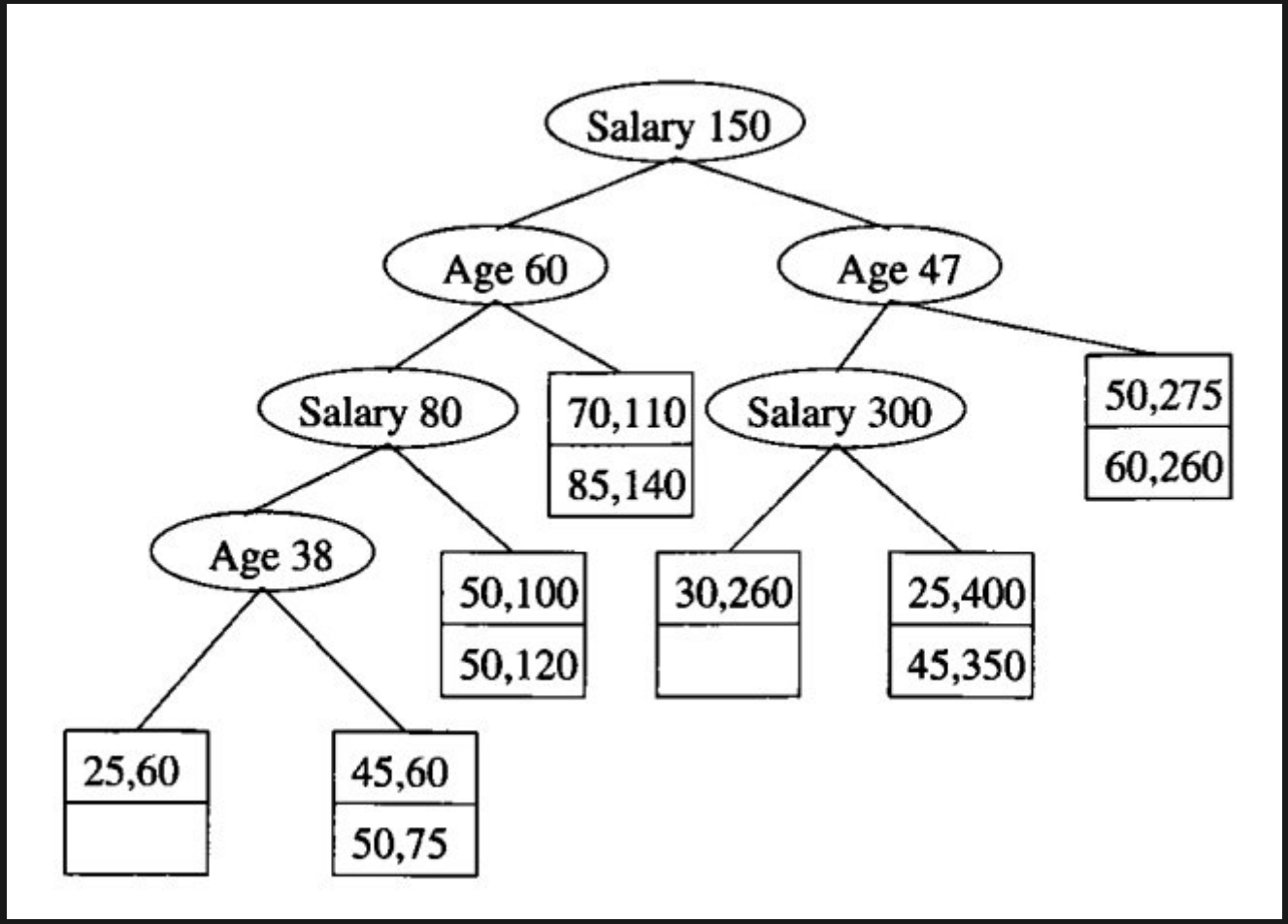
KD-ДЕРЕВЬЯ

- Посчитаем медиану по одному параметру
- Заведем корень дерева с медианой как значением
- Кто меньше - тот слева, кто больше - тот справа

KD-ДЕРЕВЬЯ

- Повторим схему рекурсивно для каждой половины
- Только по второму атрибуту
- В каждой половинке - своя медиана
- И пойдем дальше

КАРТИНКА


Salary 150
Age 60
Age 47
Salary 80
70,110
85,140
Salary 300
50,275
60,260
Age 38
50,100
50,120
30,260
25,400
45,350
25,60
45,60
50,75

ГЕОМЕТРИЧЕСКАЯ МОДЕЛЬ

- Рассекаем прямоугольник пополам
- Каждую половинку - снова пополам
- Каждую по-своему
- (В отличие от грида)

КАРТИНКА

ОЦЕНИМ СЛОЖНОСТЬ

- Сложность поиска на толстых диапазонах не так важна
- Потому что уже проблема длинной выдачи
- Или плохой локализации в случае БД
- Оценим для точных запросов

ОЦЕНИМ СЛОЖНОСТЬ

- Если запрос по обоим критериям - идем до листьев
- За логарифм или дойдем, или поймем, что ключа нет
- Если по одному - то рассматриваем несколько путей
- Через два шага от корня получим 4 узла
- Два из которых нам интересны как "кандидаты"

ОЦЕНИМ СЛОЖНОСТЬ

- Еще через два шага каждый из 4 узлов породит по 4
- В итоге - 16
- А два интересных - породят по 2 интересных
- И так далее
- Посетим и проверить квадратный корень узлов

ОСМЫСЛИМ РЕЗУЛЬТАТ

- Сублинейно, но хуже чем логарифм
- Будь у нас дерево деревьев - получили бы логарифм
- Но только по одному из ключей
- А по двойному поиску - и там, и там логарифм

УЗКИЕ ДИАПАЗОНЫ

- Почти как точные
- Но даже узкий диапазон может пересечь границу
- Тогда посмотрим лишнее поддерево
- Но если он узкий, это будет разовая ситуация
- И асимптотически неважно, на каком уровне это произойдет

ДОБАВЛЕНИЕ

- Гарантированного баланса нет
- Но есть надежда на то, что распределение добавляемых похоже на распределение уже добавленных
- Особенно если дерево создано уже на большом наборе данных
- (А на малом нам и индекс не нужен)

ДОБАВЛЕНИЕ

- Даже в in-memory версии в листьях не одиночные листы, а блоки
- По мере их наполнения они раздваиваются
- Есть шанс, что в целом равномерно
- Если накапливается перекос - переиндексируем

ПОИСК БЛИЖАЙШИХ

- Начинаем с точного поиска точки
- Ищем кандидатов в том же "прямоугольнике"
- Разбираемся, надо ли других искать
- Если надо - ищем у соседей

ПОИСК БЛИЖАЙШИХ

- Только придется пройтись до корня
- Соседи могут быть не обязательно слева
- В разумной ситуации должно быть недорого

НА ДИСКЕ

- Первый подход: слабый закос под B-дерево
- По первому измерению разбиваемм не на 2, а на много
- Аналогично по второму
- А растем - как kd, с поправкой на многодетность

НА ДИСКЕ

- Нет гарантий на одинаковую высоту
- В геометрической модели - шинкуем прямоугольник на сотни полосок
- И потом каждую на сотни
- Здесь могут быть проблемы

НА ДИСКЕ

- Диапазонные запросы могут покрывать несколько "полосок"
- После 3 уровня получаем мелкое разбиение
- На 4-й данных может не хватить
- Получается "ассиметрия"

НА ДИСКЕ

- Второй подход: уплотнение
- Берем несколько уровней kd-дерева
- Поселяем их в одном блоке
- В 4-8K храним что-то типа бинарного дерева в массиве

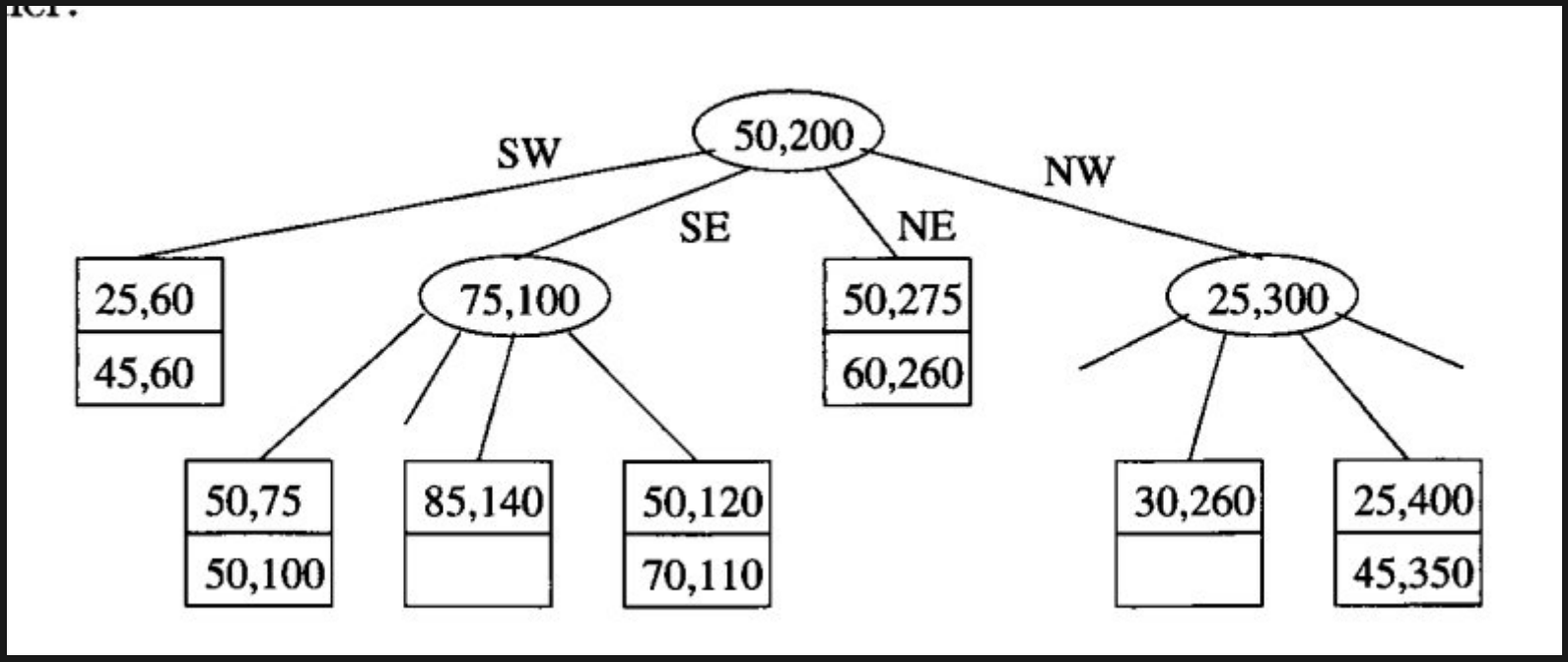
ВОЗМОЖНАЯ НЕПРИЯТНОСТЬ

- А что, если точка попадает на границу ?
- Можно подвинуть границу
- Но не всегда возможно из-за дискретности
- Или потому что многие значения попадают на границу
- База выкрутится, но эффективность может понизиться

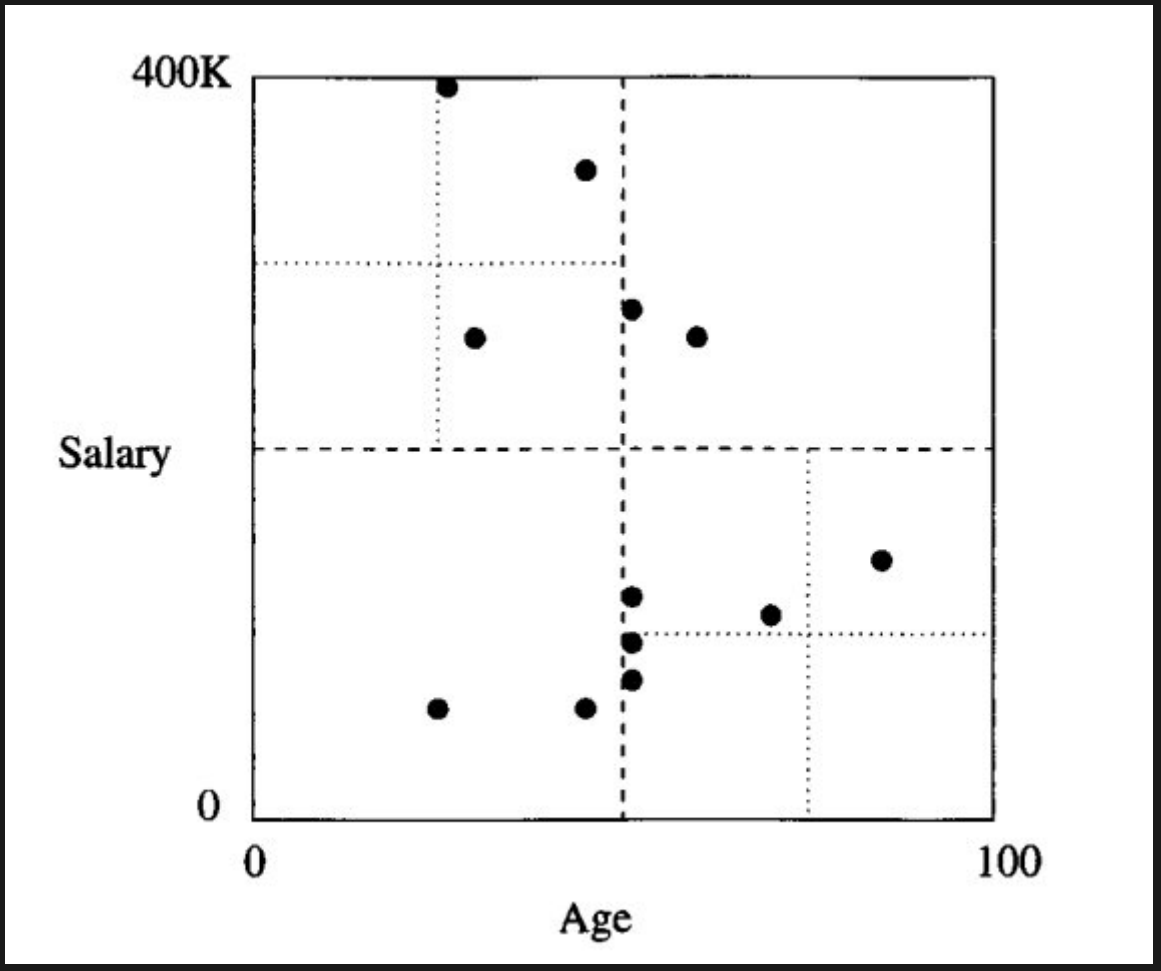
QUAD-ДЕРЕВЬЯ

- Вариация на тему kd-деревьев
- В геометрической модели берем точку в прямоугольнике
- И проводим линии, параллельные сторонам
- Делим на 4 сектора

КАРТИНКА


50,200
SW
SE
NE
NW
25,60
45,60
75,100
50,275
60,260
25,300
50,75
50,100
85,140
50,120
70,110
30,260
25,400
45,350

КАРТИНКА



ТРАНЗАКЦИИ

- Начнем с пользовательской точки зрения
- С темы изоляции
- SQL определяет 4 уровня изоляции
- В терминах аномалий, которые могут или не могут произойти

LOST UPDATE

- Одна транзакция читает запись
- Что-то вычисляет на основании прочитанного и обновляет запись
- Другая параллельно делает то же самое
- Чье-то обновление теряется
- Классический пример: параллельное увеличение баланса
- Запрещено на любых уровнях

DIRTY READ

- Одна транзакция добавляет сумму на пустой счет
- Но пока не завершилась
- Другая читает это значение
- На уровне изоляции 'Read Uncomitted' разрешены грязные чтения

DIRTY READ

- Что-то делает на основании прочитанного (оплачивает счет)
- А первая отменяется
- В итоге ушли в минус
- Чего вообще-то нельзя было делать

NON-REPEATABLE READS

- Внутри транзакции проходят несколько чтений
- Между чтениями другая транзакция завершается успешно
- Первая транзакция читает ее результат

ПРИМЕР

- Есть ограничение, запрещающее негативный баланс
- На счете $1000
- Первая транзакция проверяет сумму, читает $1000
- Другая транзакция снимает $1000 и успешно завершается

ПРИМЕР

- Первая транзакция уверена, что на счету $1000
- Вычитает $100
- Получается минус на счете
- И здесь читались только закомиченные результаты
- Уровень 'Read committed' запрещает незакомиченные чтения
- Но допускает неповторяющиеся чтения

PHANTOM READS

- Одна транзакция выполняет два одинаковых SELECT-запроса
- Между ними другая транзакция добавляет записи, удовлетворяющие критерию
- И завершается
- Первая получает разные ответы на один SELECT

PHANTOM READS

- Пример: клиенту запрещено иметь более трех счетов
- Две транзакции одновременно выполняют одну и ту же логику
- Считают количество счетов
- Если их не больше двух - создают новый
- Рискуем получить 4 счета

REPEATABLE READ

- На уровне Reapeatable Read запрещены неповторяющиеся чтения
- Но допускаются фантомные чтения
- И есть уровень Serializable

SERIALIZABLE

- Уровень, на котором запрещены аномалии
- Параллельные транзакции имеют такой эффект, как будто исполняются одна за другой

	lost update	dirty read	non-repeatable read	phantom read	other anomalies
Read Uncommitted	–	yes	yes	yes	yes
Read Committed	–	–	yes	yes	yes
Repeatable Read	–	–	–	yes	yes
Serializable	–	–	–	–	–

УРОВНИ ИЗОЛЯЦИИ И ЛОКИ

- Просматривается связь между уровнями изоляции и их реализацией на локах
- Без локов возможно что угодно
- Будем лочить обновляемые записи на обновление - получим Read Uncommitted
- Будем лочить обновляемые записи на обновление и чтение - получим Read committed

УРОВНИ ИЗОЛЯЦИИ И ЛОКИ

- Будем лочить еще и при чтении - получим Repeatable Read
- А с фантомными чтениями - проблема
- Непонятно, что бы такое там залочить
- Можем локами предотвратить удаление прочитанных

SNAPSHOT

- Никто ничего в лоб не лочит
- Используется снепшотная изоляция
- В ней есть разные подходы
- Postgres использует MVCC

РЕАЛЬНОСТЬ POSTGRES

- Грязные чтения невозможны
- Read Uncommitted - формально можно указать
- Но по факту не отличий от Read Committed
- В некоторых случаях возможны потерянные обновления

РЕАЛЬНОСТЬ POSTGRES

- В RepeatableRead нет фантомных чтений
- Это сильнее, чем стандарт
- Но есть другие аномалии
- Стандарт говорит не о всем

POSTGRES

	lost updates	dirty reads	non-repeatable reads	phantom reads	other anomalies
Read Committed	yes	–	yes	yes	yes
Repeatable Read	–	–	–	–	yes
Serializable	–	–	–	–	–