C# Developer.

Базы данных: реляционные базы и работа с ними

otus.ru

Проверить, идет ли запись

Меня хорошо видно && слышно?

Ставим “+”, если все хорошо
“-”, если есть проблемы

Герасименко Антон

Немного о себе:

- Работаю в компании “Интелком лайн” fullstack lead dev

- Код: в основном на dotnet, с 2010, typescript

- преподаватель курса “C# ASP.NET Core разработчик” и остальных .net курсов в OTUS + Базы данных

Контакты:

Правила вебинара

Активно
участвуем

Задаем вопрос
в чат или голосом

Вопросы вижу в чате,
могу ответить не сразу

Условные обозначения

Индивидуально

Время, необходимое
на активность

Пишем в чат

Говорим голосом

Документ

Ответьте себе или
задайте вопрос

Маршрут вебинара

- Оптимизация
- Индексы
- ADO.Net + Dapper
- OLTP vs OLAP
- Примеры
- Рефлексия

Цели вебинара

К концу занятия вы сможете

1.	Работать с базой с помощью Ado.Net
2	Определение производительности и оптимизация
3.	Писать запросы на micro-ORM Dapper

Смысл

Зачем вам это уметь

1. Иметь возможность выбрать технологию для работы с БД
2. Уметь читать/писать данные в БД

Производительность запросов и оптимизация

Причины медленных запросов

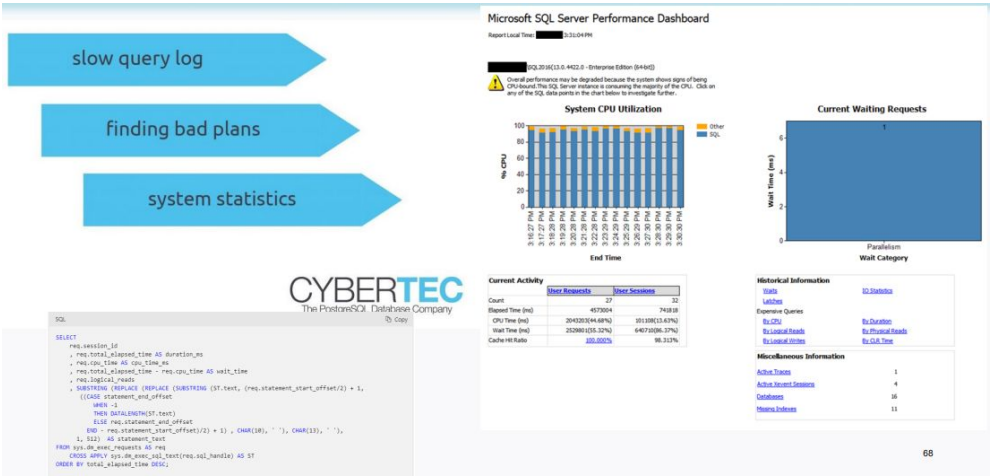
- Отсутствие индексации: Сканирует целые таблицы вместо использования индексов.
- Слишком много объединений: Избыточные или ненужные объединения увеличивают сложность.
- Большие наборы результатов: Запросы возвращают больше данных, чем необходимо.
- Плохо написанный SQL: Неэффективные запросы или плохой синтаксис могут снизить производительность.

Методы оптимизации запросов

- Индексирование: используйте правильные типы индексов для ускорения доступа к данным.
- *Избегайте SELECT: указывайте только необходимые столбцы.
- Ограничение объединений: упрощайте запросы, сокращая количество объединений.
- Используйте предложения WHERE: фильтруйте данные как можно тщательнее.
- Используйте кэширование запросов: сохраняйте часто выполняемые запросы для более быстрого доступа.

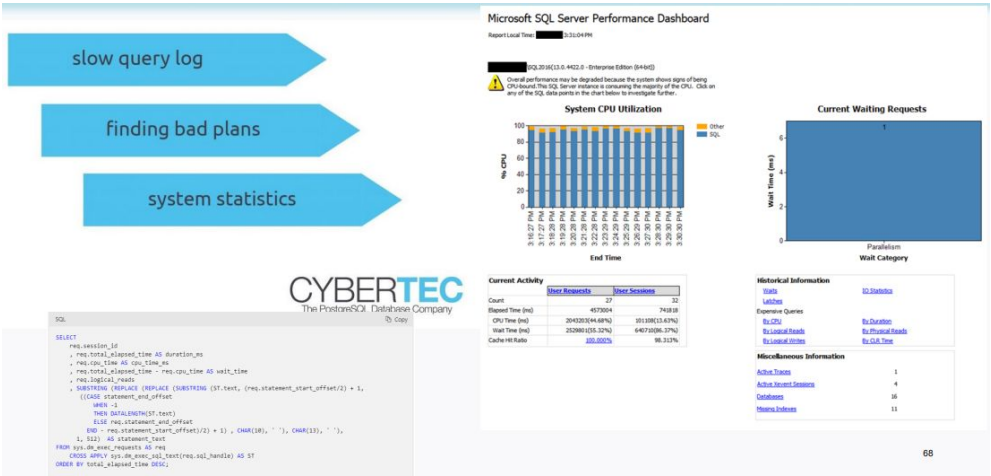
Мониторинг производительности запросов

- Используйте EXPLAIN в SQL для анализа планов выполнения запросов.
- Такие инструменты, как SQL Profiler или pgAdmin, помогают отслеживать медленные запросы.
- Отслеживайте время ответа: ищите медленно выполняющиеся запросы, которые превышают пороговые значения.
- Регулярно просматривайте журналы запросов для потенциальных оптимизаций.



Лучшие практики оптимизации запросов

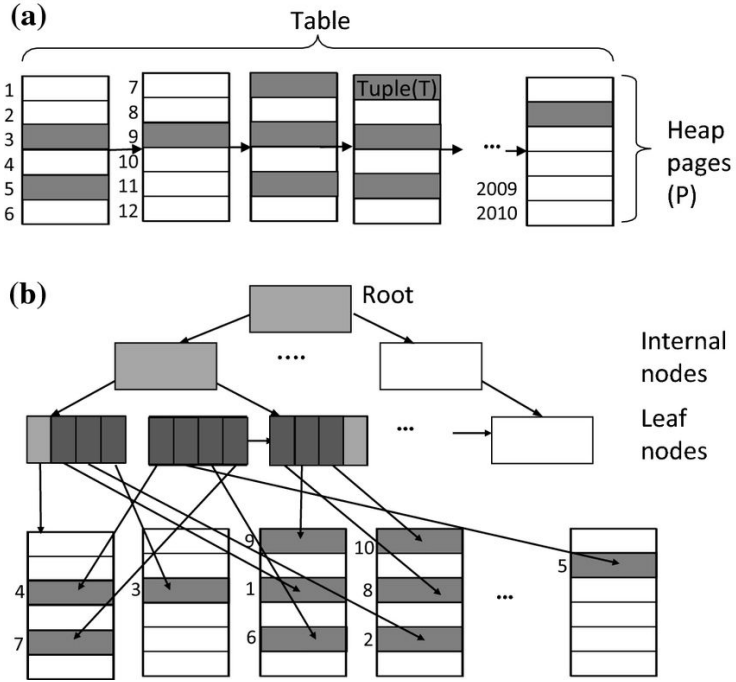
- Регулярно проверяйте и обновляйте индексы на основе использования запросов.
- Избегайте вложенных подзапросов; используйте объединения, где это возможно.
- Batch обновления и удаления для уменьшения блокировок.
- Тестируйте запросы в реальных сценариях перед развертыванием.
- Постоянно отслеживайте медленные запросы и снижение производительности.



Выбор типов индексов для оптимизации запросов

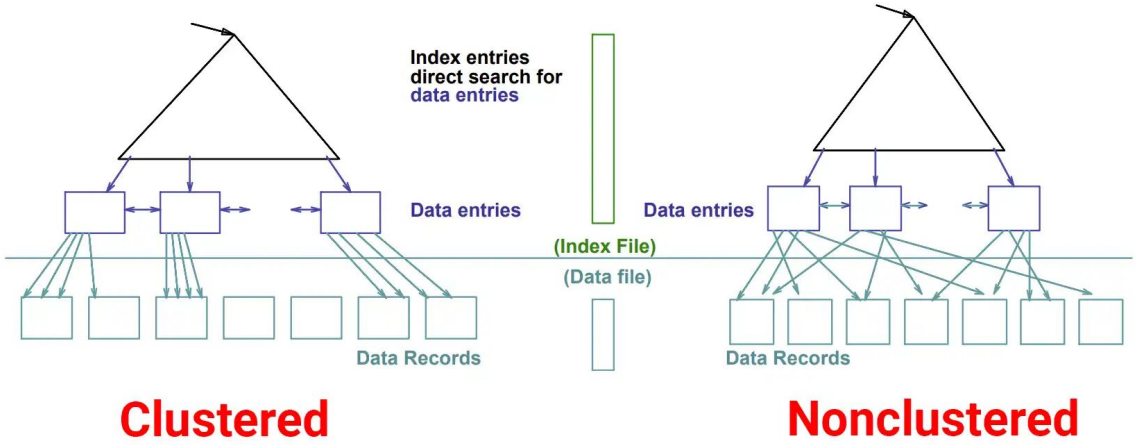
Почему индексы важны?

- Индексы позволяют быстрее извлекать данные без сканирования всей таблицы.
- Они действуют как «таблица содержания» для вашей базы данных.
- Индексы сокращают время запроса, особенно для больших наборов данных.
- Плохая индексация может замедлить вставки, обновления и удаления.



Типы индексов

- Кластеризованный индекс: Сортирует и сохраняет строки физически в таблице.
- Некластеризованный индекс: Отдельная структура от таблицы, как указатель на данные.
- Составной индекс: Индексирует по нескольким столбцам для сложных запросов.
- Уникальный индекс: Гарантирует отсутствие дубликатов значений в столбце.



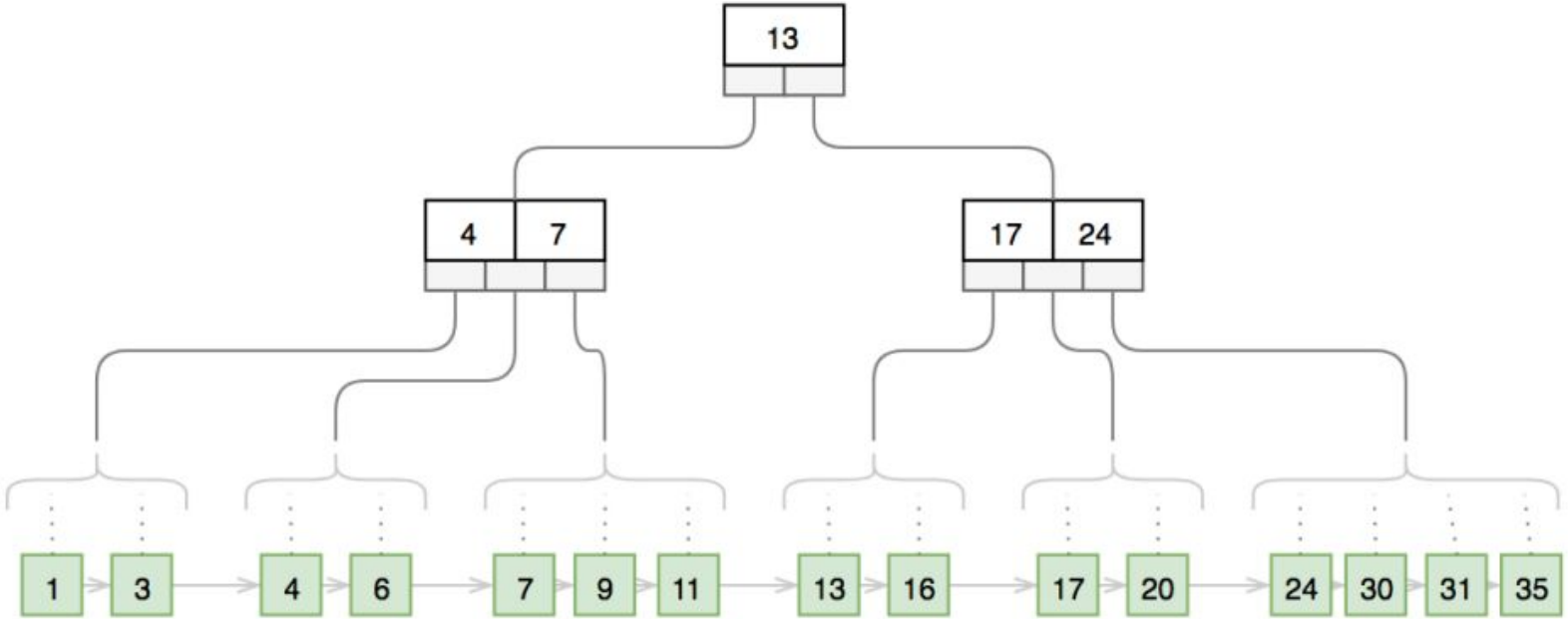
Индексы - кластеризованные

- Кластеризованные индексы сортируют и хранят записи в таблицах или представлениях на основе их ключевых значений. Этими значениями являются столбцы, включенные в определение индекса. Существует только один кластеризованный индекс для каждой таблицы, так как записи в таблице могут храниться в единственном порядке.
- Записи в таблице хранятся в порядке сортировки только в том случае, если таблица содержит кластеризованный индекс. Если у таблицы есть кластеризованный индекс, то таблица называется кластеризованной. Если у таблицы нет кластеризованного индекса, то строки данных хранятся в неупорядоченной структуре, которая называется кучей (heap file, не путать со структурой данных).

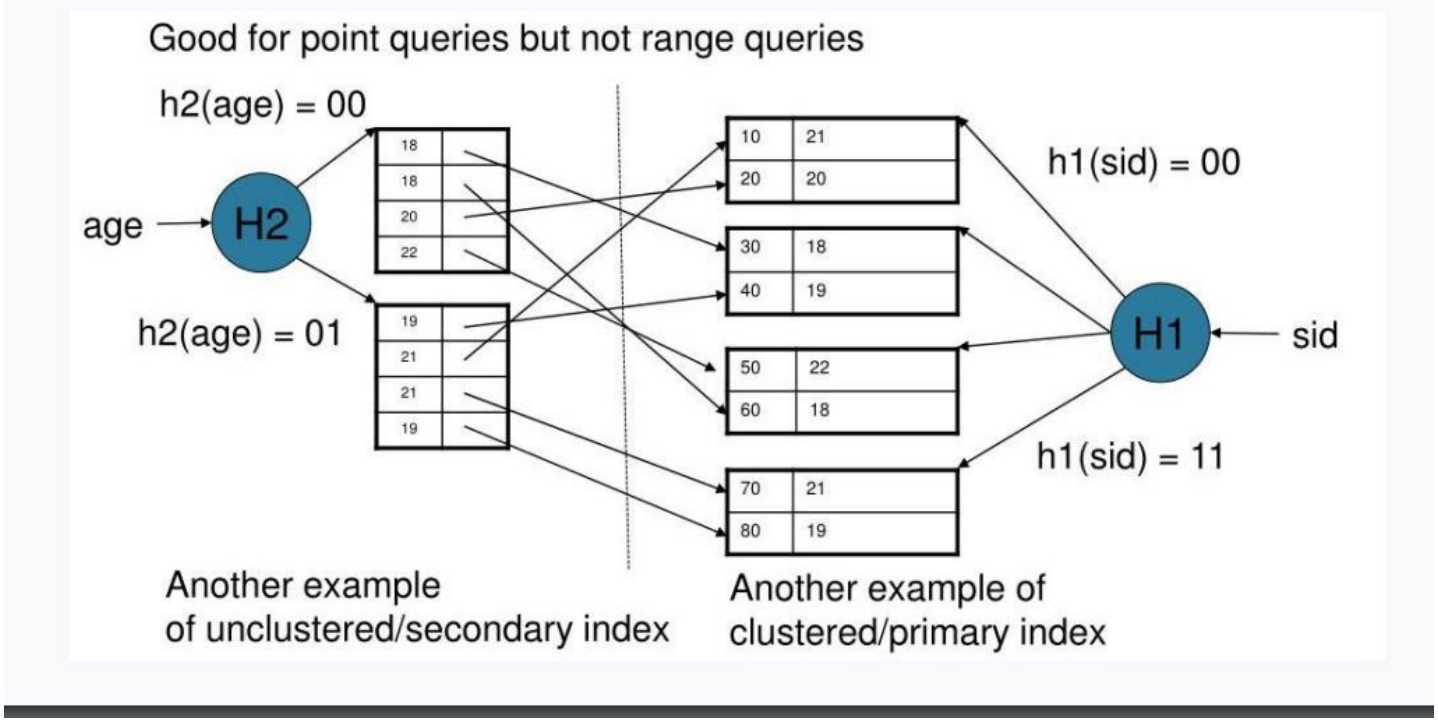
Индексы - некластеризованные

- Некластеризованные индексы имеют структуру, отдельную от основной структуры хранения таблицы. В некластеризованном индексе содержатся значения ключа некластеризованного индекса, и каждая запись значения ключа содержит указатель на строку данных, содержащую значение ключа.
- Указатель из строки индекса в некластеризованном индексе, который указывает на строку данных, называется указателем строки. Структура указателя строки зависит от того, хранятся ли страницы данных в куче или в кластеризованной таблице. Для кучи указатель строки является указателем на строку. Для кластеризованной таблицы указатель строки данных является ключом кластеризованного индекса.
- Вы можете добавить неключевые столбцы на конечный уровень некластеризованного индекса, чтобы обойти существующее ограничение на ключи индексов и выполнять полностью индексированные запросы.

B-деревья



Хеш-индексы

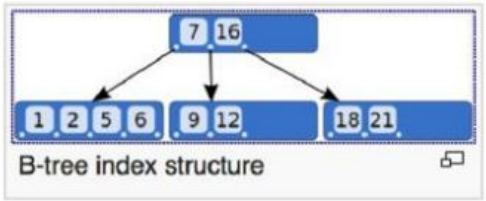

Good for point queries but not range queries
h2(age) = 00
age
H2
h2(age) = 01
18
18
20
22
19
21
21
19
10
21
20
20
30
18
40
19
50
22
60
18
70
21
80
19
h1(sid) = 00
H1
sid
h1(sid) = 11
Another example
of unclustered/secondary index
Another example of
clustered/primary index

Block Range Index (BRIN)

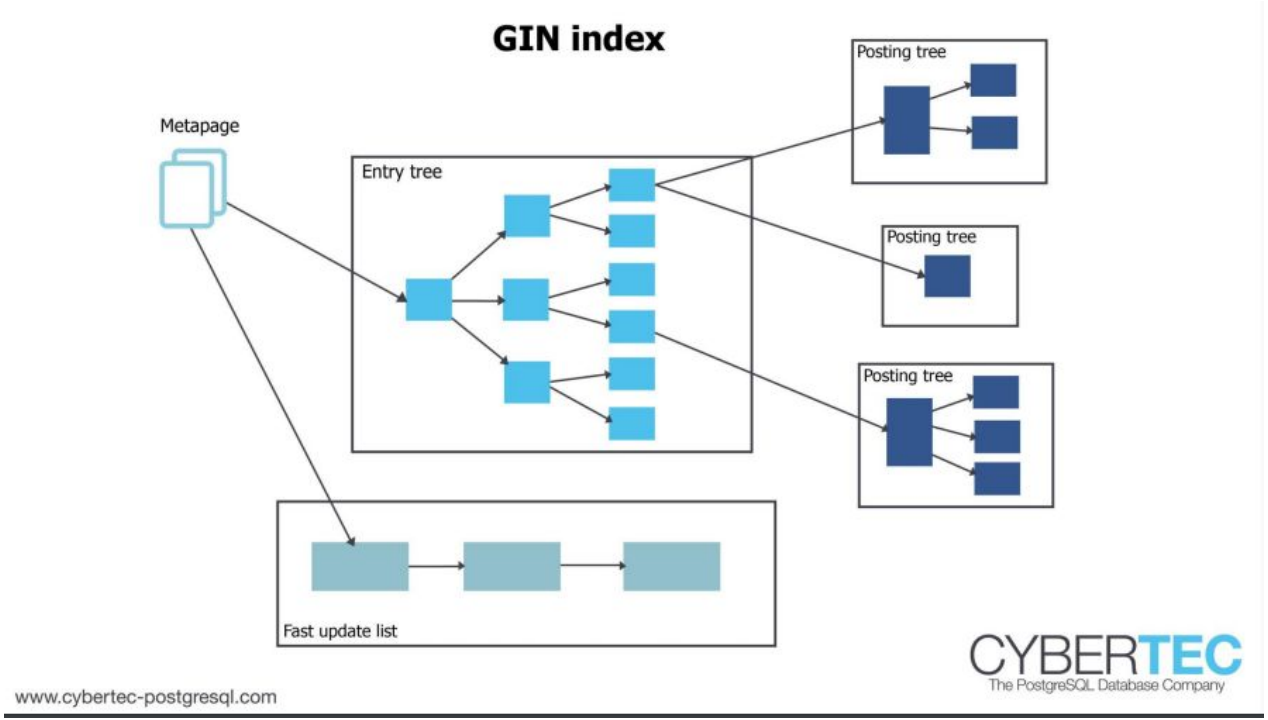
```
CREATE INDEX brin_index_test  ON test_data
   USING brin(id)
   WITH (pages_per_range = 128);
```

Lossy!

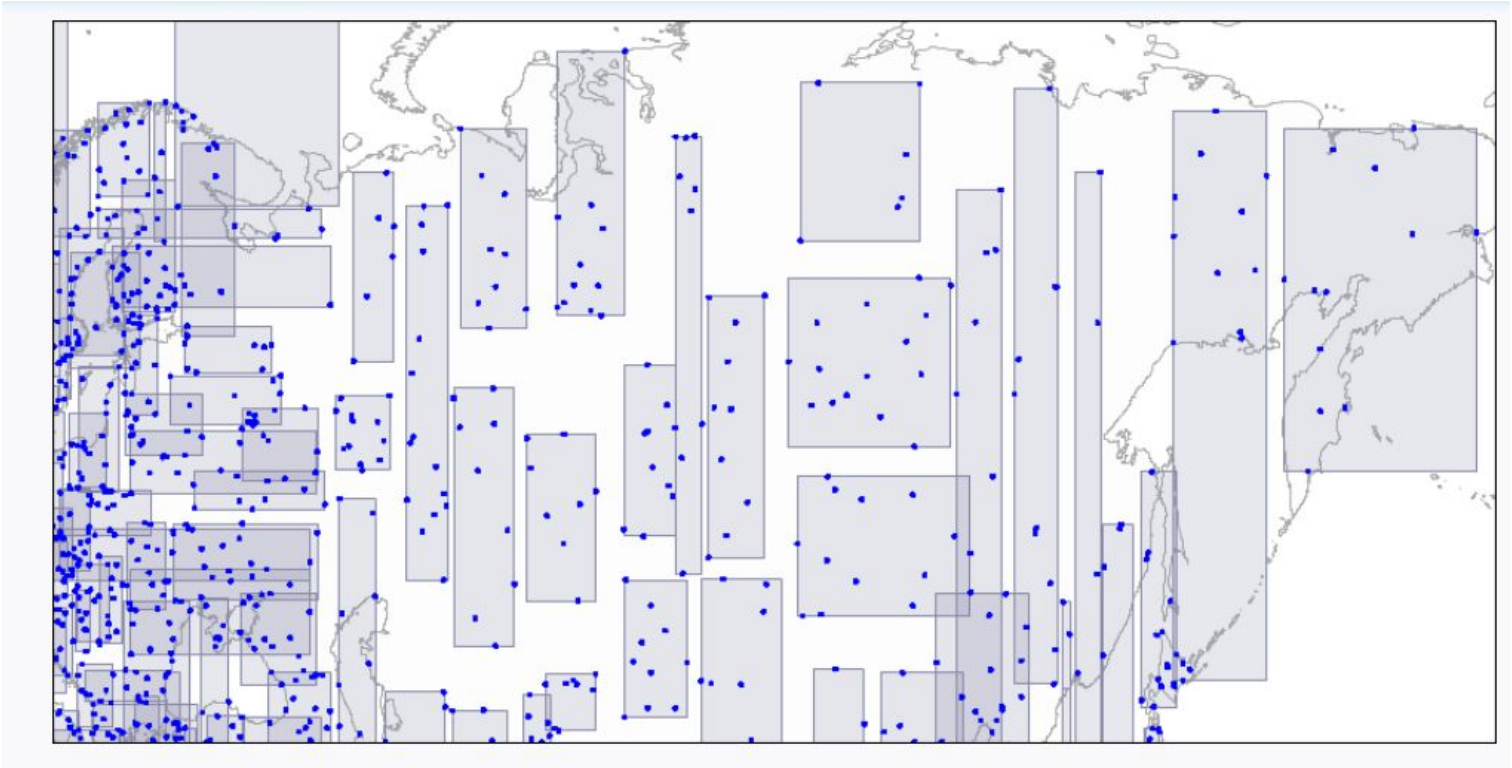
- Индексируются не записи, а страницы. Внутри страницы — sequence scan
- Хорошо подходит для append-only таблиц
- Быстро строится и занимает очень мало места
- Ищет медленнее, чем B-Tree



GIN


GIN index
Metapage
Entry tree
Posting tree
Posting tree
Posting tree
Fast update list
www.cybertec-postgresql.com
CYBERTEC
The PostgreSQL Database Company

GIST

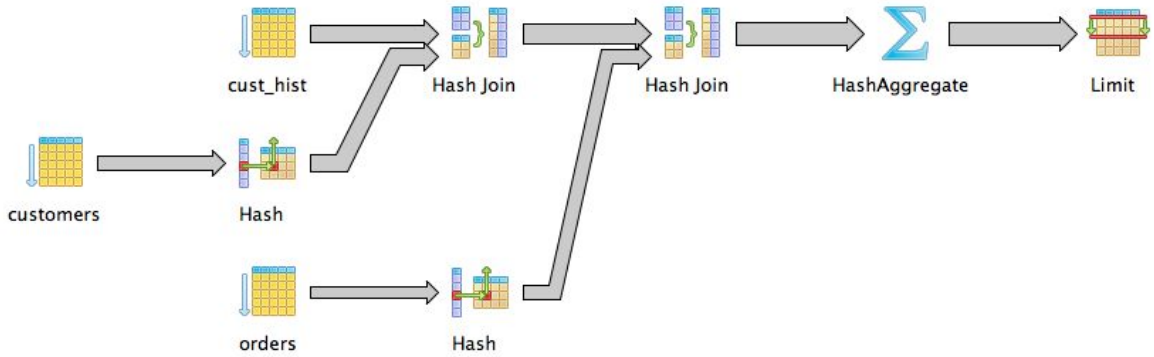


Лучшие практики индексирования

- Индексируйте селективные столбцы: столбцы с высокой степенью изменчивости (например, идентификаторы клиентов).
- Избегайте индексирования небольших столбцов с низкой степенью изменчивости (например, пол, логические значения).
- Не индексируйте чрезмерно: слишком много индексов замедляют операции записи.
- Регулярно отслеживайте и перестраивайте индексы по мере изменения данных.

Работа с планом запроса

- Sequential Scan
- Index seek
- Index Scan
  - Index Scan
  - Index Only Scan
  - Bitmap Index Scan
- JOINs
  - Nested Loop
  - Hash Join
  - Merge Join
- Parallel Nodes
  - Gather
  - Merge




```
QUERY PLAN
Sort  (cost=169.51..172.01 rows=1000 width=87) (actual time=5.098..5.138 rows=1000 loops=1)
  Sort Key: f.title
  Sort Method: quicksort  Memory: 103kB
  ->  Hash Join  (cost=41.93..119.68 rows=1000 width=87) (actual time=0.541..1.054 rows=1000 loops=1)
        Hash Cond: (f.film_id = fc.film_id)
        ->  Seq Scan on film f  (cost=0.00..64.00 rows=1000 width=19) (actual time=0.013..0.282 rows=1000 loops=1)
        ->  Hash  (cost=29.43..29.43 rows=1000 width=70) (actual time=0.515..0.515 rows=1000 loops=1)
              Buckets: 1024  Batches: 1  Memory Usage: 49kB
              ->  Hash Join  (cost=1.36..29.43 rows=1000 width=70) (actual time=0.038..0.350 rows=1000 loops=1)
                    Hash Cond: (fc.category_id = c.category_id)
                    ->  Seq Scan on film_category fc  (cost=0.00..16.00 rows=1000 width=4) (actual time=0.010..0.134 rows=1000 loops=1)
                    ->  Hash  (cost=1.16..1.16 rows=16 width=72) (actual time=0.018..0.018 rows=16 loops=1)
                          Buckets: 1024  Batches: 1  Memory Usage: 9kB
                          ->  Seq Scan on category c  (cost=0.00..1.16 rows=16 width=72) (actual time=0.010..0.012 rows=16 loops=1)
Planning time: 0.393 ms
Execution time: 5.271 ms
```

Join, Group by

Inner Join

- Объединяет столбцы двух таблиц по общему признаку
- Если не удалось сопоставить – в результирующую выборку не попадает

```
SELECT p.id, p.name, ps.id, ps.name
FROM persons p
~~INNER~~ JOIN positions ps ON ps.id = p.post_id
```

persons

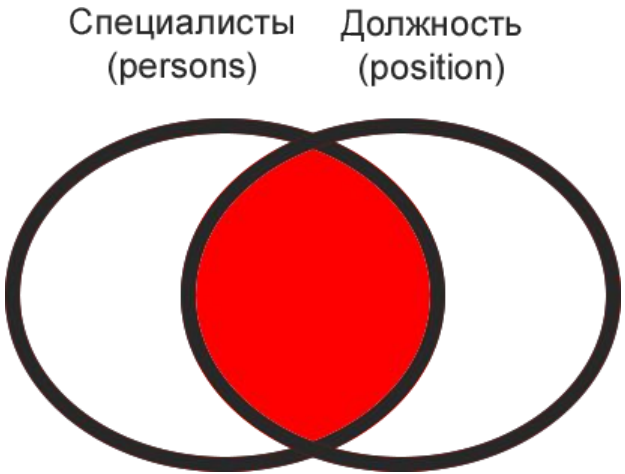
id	name	post_id
1	Владимир	1
2	Татьяна	2
3	Александр	6
4	Борис	2

positions

id	name
1	Дизайнер
2	Редактор
3	Программист

результат джоина

id	Имя сотрудника	pos.id	Должность
1	Владимир	1	Дизайнер
2	Татьяна	2	Редактор
4	Борис	2	Редактор



Left Join

- Объединяет столбцы двух таблиц по общему признаку
- Если не удалось сопоставить – по всем выбранным столбцам правой таблицы подставляется null

```
SELECT p.id, p.name, ps.id, ps.name
FROM persons p
LEFT JOIN positions ps ON ps.id = p.post_id
```

persons

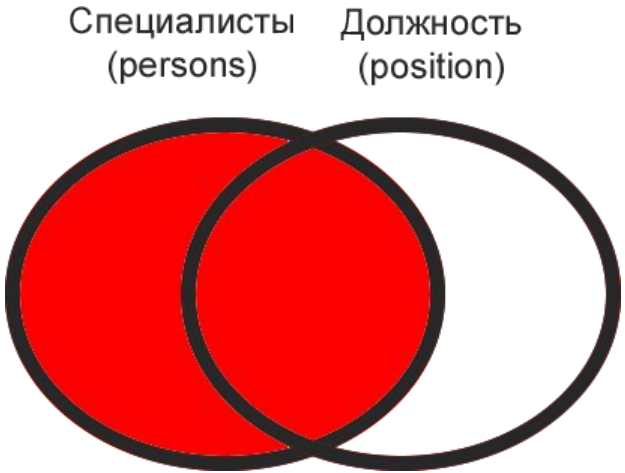
id	name	post_id
1	Владимир	1
2	Татьяна	2
3	Александр	6
4	Борис	2

positions

id	name
1	Дизайнер
2	Редактор
3	Программист

результат джоина

id	Имя сотрудника	pos.id	Должность
1	Владимир	1	Дизайнер
2	Татьяна	2	Редактор
4	Борис	2	Редактор
3	Александр	*NULL*	*NULL*



Right Join

- То же, что и Left Join, но порядок таблиц другой
- Обычно вместо него используют Left Join, поменяв таблицы местами

```
SELECT p.id, p.name, ps.id, ps.name
FROM persons p
RIGHT JOIN positions ps ON ps.id = p.post_id
```

persons

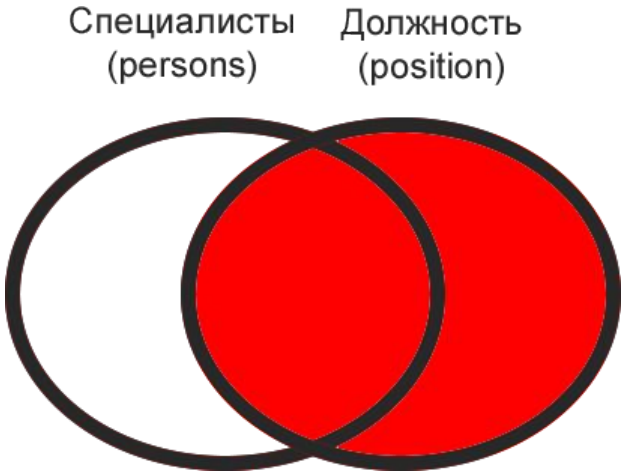
id	name	post_id
1	Владимир	1
2	Татьяна	2
3	Александр	6
4	Борис	2

positions

id	name
1	Дизайнер
2	Редактор
3	Программист

результат джоина

id	Имя сотрудника	pos.id	Должность
1	Владимир	1	Дизайнер
2	Татьяна	2	Редактор
4	Борис	2	Редактор
NULL	*NULL*	3	Программист



Left Outer Join

- Объединяет столбцы двух таблиц по различающемуся признаку
- В выборку попадают только те строки левой таблицы, для которых нет сопоставления из второй

```
SELECT p.id, p.name, ps.id, ps.name
FROM persons p
LEFT JOIN positions ps ON ps.id = p.post_id
WHERE ps.id IS NULL
```

persons

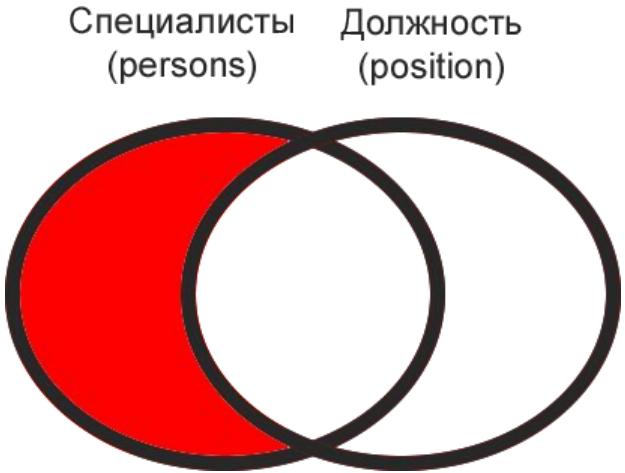
id	name	post_id
1	Владимир	1
2	Татьяна	2
3	Александр	6
4	Борис	2

positions

id	name
1	Дизайнер
2	Редактор
3	Программист

результат джоина

id	Имя сотрудника	pos.id	Должность
3	Александр	*NULL*	*NULL*



Другие Join

- Другие виды джоинов используются редко

Group By + Having

- Group By группирует выборку по признакам
- Having – похож на Where, но применяется к результату группировки
- Where нельзя использовать с агрегирующими функциями

Group By + Having

- Group By группирует выборку по признакам
- Having – похож на Where, но применяется к результату группировки
- Where нельзя использовать с агрегирующими функциями

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5
ORDER BY COUNT(CustomerID) DESC;
```

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

COUNT(CustomerID)	Country
13	USA
11	Germany
11	France
9	Brazil
7	UK

ADO.NET
Что это?

ADO.NET

ADO.NET по-прежнему остается актуальной и широко используемой технологией доступа к данным в приложениях .NET.

Он предоставляет набор классов для автономного доступа к источникам данных, таким как базы данных и XML-файлы.

ADO.NET является частью .NET Framework и .NET Core/.NET 5+ (теперь называемой просто .NET) и остается основополагающей технологией доступа к данным. Хотя появились новые технологии и платформы доступа к данным, такие как Entity Framework (EF) и Dapper, ADO.NET далеко не устарел.

ADO.NET

Производительность: ADO.NET предлагает детальный контроль над взаимодействием с базой данных, который может быть более эффективным для определенных сценариев с высокой производительностью по сравнению с инструментами ORM (объектно-реляционное сопоставление), такими как Entity Framework.

Гибкость: он предоставляет возможность выполнять необработанные SQL-запросы, хранимые процедуры и обрабатывать сложные транзакции, предлагая больший контроль над операциями с базой данных.

ADO.NET

Легкость: для приложений, не требующих накладных расходов на ORM, ADO.NET может быть более простым и понятным выбором.

Совместимость: ADO.NET совместим с широким спектром поставщиков данных, что делает его универсальным для доступа к различным типам баз данных.

Зрелая и стабильная. Будучи зрелой технологией, ADO.NET имеет хорошо зарекомендовавший себя и стабильный API с обширной документацией и поддержкой сообщества.

ADO.NET

1.	SqlConnection	Подключение к БД с использованием Connection String
2	SqlCommand	Команда (обертка над запросом) для отправки на сервер Позволяет также подставлять параметры в запрос
3.	SqlTransaction	Транзакция для выполнения нескольких запросов как единого целого
4.	SqlDataReader	Удобное средство для чтения множества строк из БД
5.	DataSet	Редко используемое (узкоспециализированное) средство для чтения таблицы БД и оперирования с ней оффлайн

ADO.NET

ExecuteScalar — это метод в ADO.NET, который используется для выполнения запроса SQL и возврата первого столбца первой строки в наборе результатов, возвращаемом запросом. Любые другие столбцы и строки игнорируются. Этот метод обычно используется, когда вы хотите получить одно значение из базы данных, например число, сумму или значение определенного столбца из одной записи.

```
ссылок: 0
public static async Task<int> GetUserCountAsync(string connectionString)
{
    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        await connection.OpenAsync();
        string query = "SELECT COUNT(*) FROM Users";

        using (SqlCommand command = new SqlCommand(query, connection))
        {
            // ExecuteScalar returns the first column of the first row in the result set
            object result = await command.ExecuteScalarAsync();
            if (result != null)
            {
                return Convert.ToInt32(result);
            }
            return 0;
        }
    }
}
```

ADO.NET async

SqlConnection.OpenAsync()
SqlCommand.ExecuteNonQueryAsync()
SqlCommand.ExecuteReaderAsync()
SqlCommand.ExecuteScalarAsync()
SqlDataReader.ReadAsync()

Преимущества и недостатки ADO.NET

+- Использование чистого SQL

+ Полный контроль за отправляемыми запросами

- Конвертация “C#-объект <-> SQL-представление” целиком на разработчике

- IDE не поможет с переименованиями

- Компилятор не поможет с контролем типов

Когда использовать ADO.NET

Сложные транзакции, требующие детального контроля над соединением, командами и транзакциями.

Ситуации, когда разработчику необходимо использовать расширенные функции поставщика базы данных.

Обширных пользовательских манипуляций с данными и тонкой настройки производительности на уровне взаимодействия с базой данных.

Минимальные внешние зависимости, поскольку ADO.NET является частью .NET Framework.

Пример

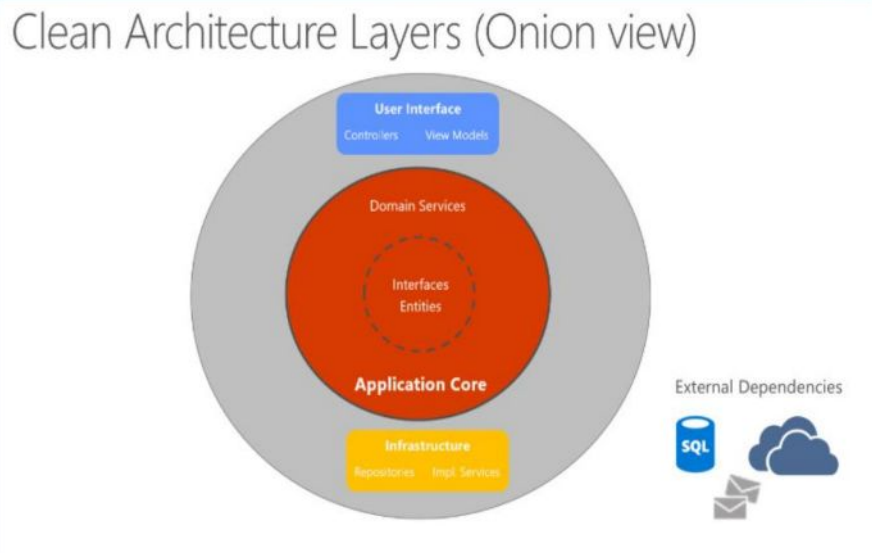
ORM.
Что это?

ORM

ORM (Object-Relational Mapping): объектно-реляционное отображение — технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных».

ORM объединяет по своей сути различные парадигмы объектно-ориентированного программирования, где сущности представлены в виде классов и объектов, и реляционных баз данных, где данные хранятся в таблицах со строками и столбцами

Какие модели маппит ORM

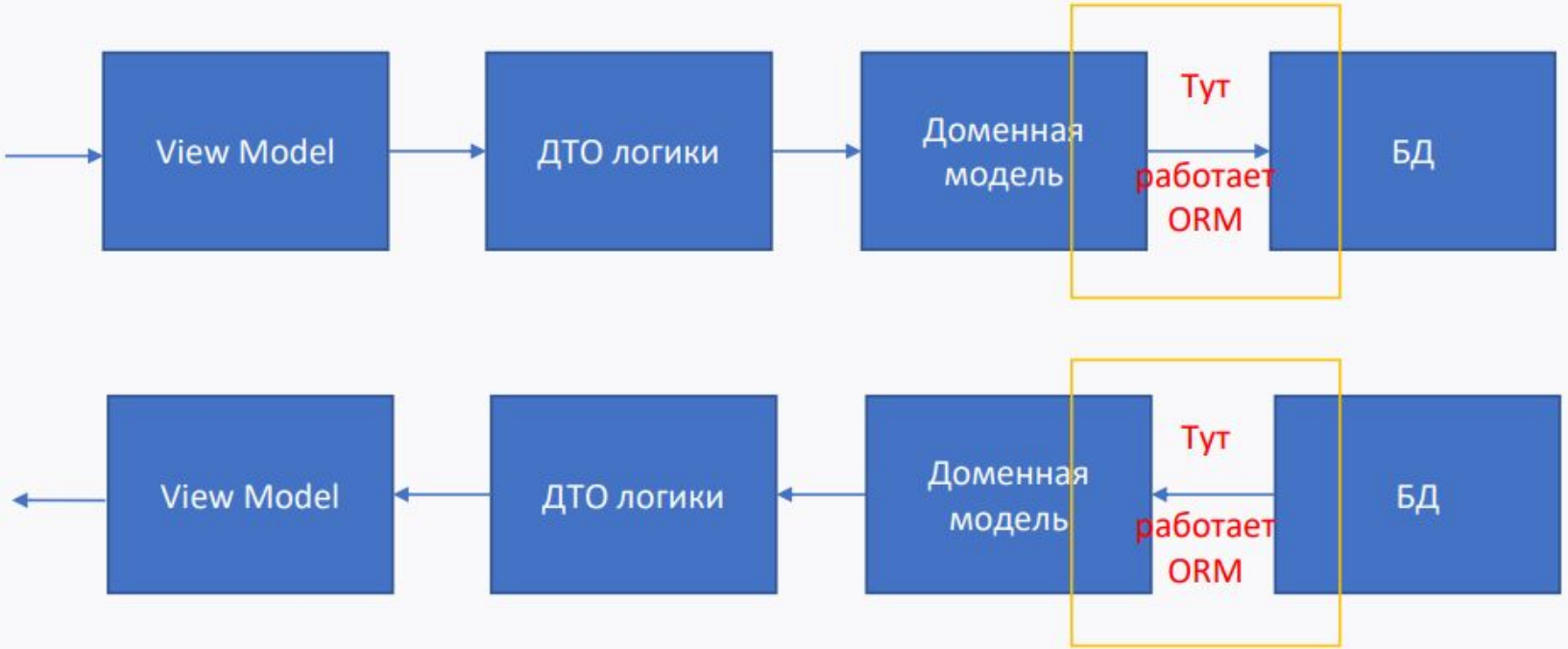


Виды моделей

1. Доменные модели (Entities), находятся в ядре
2. Модели логики (DTO), находятся в Domain Services
3. Модели для работы с клиентом (View Models), находятся в API или слое пользовательского интерфейса

К БД мапим Entities

Какие модели маппит ORM



Преимущества и недостатки ORM

Преимущества

- **Простота**: ORM предоставляет простой интерфейс для работы с базой данных, который может быть понятным любому программисту. ORM скрывает сложности SQL-запросов, позволяя работать с данными на более высоком уровне абстракции.
- **Переносимость**: ORM может работать с различными СУДБ, что делает его более переносимым, чем SQL. Это позволяет разработчикам легко переносить свое приложение на другую СУБД без изменения кода.
- **Сопровождаемость**: ORM может значительно упростить сопровождение приложения, так как изменения в структуре базы данных могут быть внесены непосредственно в код ORM, а не в каждый SQL-запрос.
- **Безопасность**: ORM может предотвратить SQL-инъекции, поскольку ORM автоматически экранирует данные, которые передаются в базу данных.

Преимущества и недостатки ORM

Недостатки

- **Сложность**: ORM может быть сложным для понимания, особенно для новых разработчиков. ORM требует определенных знаний и опыта, чтобы использовать его эффективно.
- **Производительность**: ORM может быть менее эффективным, чем работа с SQL напрямую. ORM должен обрабатывать запросы и преобразовывать их в SQL, что может замедлить производительность.
- **Ограничения**: ORM может иметь ограничения в отношении того, какие запросы могут быть выполнены. В случае, когда нужно выполнить сложный запрос или использовать специфичные функции базы данных, может потребоваться написание SQL-запроса напрямую.

Dapper

Что такое Dapper

Легкий микро-ORM, разработанный Stack Overflow, известный своей простотой и производительностью. Позволяет разработчикам выполнять необработанные SQL-запросы и сопоставлять результаты запросов с объектами с минимальными издержками.

Практика

Преимущества Dapper

- быстрый
- позволяет использовать хранимые процедуры
- видно какой именно SQL запрос вы исполняете

Сценарии использования Dapper

- приложения, критичные к производительности, которым требуется детальный контроль над взаимодействием с базой данных.
- когда требуются сложные запросы или оптимизация базы данных

OLAP
OLTP

Виды БД по назначению

- OLTP
  - OnLine Transaction Processing
  - Для оперативного учета: удобной вставки/обновления данных

Виды БД по назначению

- OLTP
  - OnLine Transaction Processing
  - Для оперативного учета: удобной вставки/обновления данных
- OLAP
  - OnLine Analytical Processing
  - Для аналитики собранных данных

Виды БД по назначению

- OLTP
  - OnLine Transaction Processing
  - Для оперативного учета: удобной вставки/обновления данных
- OLAP
  - OnLine Analytical Processing
  - Для аналитики собранных данных
- Если смешивать два этих назначения в одной БД, то
  - Вставка может быть
  - медленной (много индексов)
  - неудобной (таблицы могут быть заточены под аналитику)
- Построение аналитических отчетов может быть
  - медленным (мало индексов)
  - сложным (структура таблиц заточена под вставку, сложная для анализа)
  - затормаживать работу базы данных для вставки актуальных данных
- Часто данные из OLTP БД реплицируются (копируются) в OLAP БД, где их уже спокойно анализируют аналитики.

Тестирование

Список материалов для изучения

1. https://www.learndapper.com/
2. https://linq2db.github.io/
3. https://learn.microsoft.com/enus/dotnet/api/system.data.sqlclient.sqlconnection.connectionstring?view=dotnet-plat-ext7.0&viewFallbackFrom=net-6.0
4. https://martinfowler.com/eaaCatalog/repository.html
5. https://learn.microsoft.com/en-us/ef/core/modeling/inheritance
6. https://www.entityframeworktutorial.net/efcore/configure-one-to-one-relationship-using-fluent-api-inef-core.aspx
7. https://www.entityframeworktutorial.net/efcore/configure-one-to-many-relationship-using-fluent-apiin-ef-core.aspx
8. https://www.entityframeworktutorial.net/efcore/configure-many-to-many-relationship-in-ef-core.aspx
9. Мартин Фаулер. Архитектура корпоративных программных приложений

Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Рефлексия

Цели вебинара

Проверка достижения целей

1. Назвать и объяснить основные принципы ФП
2. Писать функциональный код на C#

Рефлексия

С какими впечатлениями уходите с вебинара?

Как будете применять на практике то,
что узнали на вебинаре?