Лекция 8

Особенности подключения объектов базы данных. Индексация

Цель

Изучить механизмы подключения к базам данных и методы индексации для оптимизации производительности, понять различные типы индексов и их практическое применение.

Основные вопросы

1. Архитектура подключения к базам данных
2. Технологии доступа к данным: ODBC, JDBC, OLE DB, ADO.NET
3. Понятие индексации и ее влияние на производительность
4. Типы индексов: кластерные, некластерные, составные, покрывающие
5. Стратегии создания и управления индексами
6. Мониторинг и обслуживание индексов

Лекция

Архитектура подключения к базам данных

Подключение к БД - процесс установления соединения между клиентским приложением и сервером базы данных для обмена данными.

Компоненты архитектуры подключения:

- Клиентское приложение;
- Драйвер или провайдер данных;
- Сетевой протокол;
- Сервер базы данных;
- База данных;

Технологии доступа к данным

ODBC (Open Database Connectivity)

ODBC - стандартный интерфейс для доступа к различным СУБД.

Архитектура ODBC:

- Приложение → Диспетчер драйверов ODBC → Специфичный драйвер ODBC → СУБД

Преимущества:

- Кроссплатформенность;
- Поддержка различных СУБД;
- Стандартизированный интерфейс;

Пример строки подключения ODBC:

```
Driver={SQLServer};Server=myServerAddress;Database=myDataBase;Uid=myUsername;Pwd=myPassword;
```

JDBC (Java Database Connectivity)

JDBC - API для доступа к базам данных из Java-приложений.

Компоненты JDBC:

- JDBC API - интерфейсы для приложений;
- JDBC Driver Manager - управление драйверами;
- JDBC Drivers - специфичные драйверы для СУБД;

Типы драйверов JDBC:

- Тип 1: JDBC-ODBC мост;
- Тип 2: Нативный API частично Java;
- Тип 3: Сетевой протокол;
- Тип 4: Нативный протокол (рекомендуется);

Пример подключения JDBC:

```
String url = "jdbc:sqlserver://localhost:1433;databaseName=MyDB";
String user = "username";
String password = "password";


Connection conn = DriverManager.getConnection(url, user, password);
```

OLE DB и ADO.NET

OLE DB - компонентная технология доступа к данным от Microsoft.

ADO.NET - эволюция OLE DB для .NET Framework.

Основные компоненты ADO.NET:

- Connection - управление подключением
- Command - выполнение команд;
- DataReader - последовательное чтение данных;
- DataAdapter - мост между БД и DataSet;
- DataSet - кэш данных в памяти;

Пример подключения ADO.NET:

```
string connectionString = "Server=localhost;Database=MyDB;User Id=user;Password=pass;";
using (SqlConnection connection = new SqlConnection(connectionString))
{
    connection.Open();
    // Работа с базой данных

}
```

Введение в индексацию

Индекс - структура данных, которая ускоряет операции извлечения данных за счет дополнительных затрат на хранение и обслуживание.

Принцип работы индекса:

- Создает отсортированную копию части данных;
- Содержит ключевые значения и указатели на строки;
- Позволяет быстро находить данные без полного сканирования таблицы;

Аналогия: Оглавление в книге

Типы индексов

Кластерные индексы (Clustered Index)

- Определяет физический порядок данных в таблице;
- В таблице может быть только один кластерный индекс;
- Данные хранятся в отсортированном порядке по ключу индекса;

Создание кластерного индекса:

```
-- SQL Server
CREATE CLUSTERED INDEX IX_Employees_EmployeeID
ON Employees (EmployeeID);

-- PostgreSQL (кластерный индекс через CLUSTER)
CREATE INDEX IX_Employees_EmployeeID ON Employees (EmployeeID);

CLUSTER Employees USING IX_Employees_EmployeeID;
```

Рекомендации для кластерных индексов:

- Создавать на столбцах с уникальными значениями;
- Предпочтительно на монотонно возрастающих значениях;
- Избегать столбцов с частыми изменениями;

Некластерные индексы (Non-Clustered Index)

- Создают отдельную структуру данных;
- Содержат ключевые значения и указатели на строки;
- В таблице может быть несколько некластерных индексов;

Создание некластерного индекса:

```
CREATE NONCLUSTERED INDEX IX_Employees_LastName
ON Employees (LastName);

-- С включением дополнительных столбцов
CREATE NONCLUSTERED INDEX IX_Employees_Department_Salary
ON Employees (DepartmentID)

INCLUDE (Salary, HireDate);
```

Составные индексы (Composite Index)

- Индекс по нескольким столбцам;
- Порядок столбцов важен для эффективности;

```
CREATE INDEX IX_Employees_DepartmentID_LastName
ON Employees (DepartmentID, LastName);
```

Правила составных индексов:

- Первый столбец должен иметь высокую селективность;
- Порядок столбцов должен соответствовать запросам;
- Избегать избыточных индексов;

Уникальные индексы (Unique Index)

- Гарантируют уникальность значений в индексируемых столбцах;

```
CREATE UNIQUE INDEX IX_Employees_Email
ON Employees (Email);

-- Уникальный составной индекс
CREATE UNIQUE INDEX IX_Employees_Department_EmployeeCode
ON Employees (DepartmentID, EmployeeCode);
```

Покрывающие индексы (Covering Index)

- Содержат все столбцы, необходимые для запроса;
- Позволяют выполнить запрос без обращения к таблице;

```
CREATE INDEX IX_Employees_Covering_Search
ON Employees (DepartmentID, HireDate)
INCLUDE (FirstName, LastName, Salary);
```

Специальные типы индексов

Filtered Indexes (SQL Server)

- Индекс по подмножеству данных;
- Уменьшает размер индекса и улучшает производительность;

```
CREATE INDEX IX_Employees_ActiveOnly
ON Employees (DepartmentID)
WHERE IsActive = 1;
```

Columnstore Indexes

- Оптимизированы для хранилищ данных и аналитических запросов
- Эффективное сжатие и пакетная обработка

```
CREATE COLUMNSTORE INDEX IX_Orders_Columnstore
ON Orders (OrderID, CustomerID, ProductID, Quantity, Price, OrderDate);
```

Full-Text Indexes

- Для полнотекстового поиска;
- Поддержка морфологии и ранжирования;

```
CREATE FULLTEXT INDEX ON Products (ProductName, Description)
KEY INDEX PK_Products;
```

Стратегии создания индексов

Анализ запросов

Идентификация кандидатов для индексации:

- Столбцы в WHERE, JOIN, ORDER BY, GROUP BY;
- Часто используемые предикаты;
- Столбцы с высокой селективностью;

Пример анализа:

```
-- Поиск медленных запросов
SELECT
   query_text = SUBSTRING(qt.text, (qs.statement_start_offset/2)+1,
      ((CASE qs.statement_end_offset
        WHEN -1 THEN DATALENGTH(qt.text)
       ELSE qs.statement_end_offset
       END - qs.statement_start_offset)/2)+1),
   execution_count = qs.execution_count,
   avg_elapsed_time = qs.total_elapsed_time / qs.execution_count
FROM sys.dm_exec_query_stats qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) qt
ORDER BY avg_elapsed_time DESC;
```

Принципы проектирования индексов

1. Баланс между чтением и записью
    - Индексы ускоряют SELECT, но замедляют INSERT/UPDATE/DELETE
2. Селективность индекса
    - Высокая селективность = высокая эффективность
    - Избегать индексов на столбцах с малым количеством уникальных значений
3. Порядок столбцов в составных индексах
    - От наиболее селективного к наименее селективному
4. Включение часто запрашиваемых столбцов
    - Использование INCLUDE для покрывающих индексов

Создание и управление индексами

Синтаксис создания индексов

```
-- Базовый синтаксис
CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED] INDEX index_name
ON table_name (column1 [ASC|DESC], column2 [ASC|DESC], ...)
[INCLUDE (include_column1, include_column2, ...)]
[WHERE filter_condition]
[WITH (index_option = value, ...)];

-- Примеры
CREATE INDEX IX_Orders_CustomerDate
ON Orders (CustomerID, OrderDate DESC)
INCLUDE (TotalAmount);

CREATE UNIQUE INDEX IX_Products_Code
ON Products (ProductCode)
WHERE ProductCode IS NOT NULL;
```

Изменение и удаление индексов

```
-- Переименование индекса
EXEC sp_rename 'Employees.IX_OldName', 'IX_NewName', 'INDEX';

-- Отключение индекса
```

```
ALTER INDEX IX_Employees_LastName ON Employees DISABLE;

-- Включение индекса
ALTER INDEX IX_Employees_LastName ON Employees REBUILD;

-- Удаление индекса

DROP INDEX IX_Employees_LastName ON Employees;
```

Мониторинг и обслуживание индексов

Анализ использования индексов

```
-- Статистика использования индексов

SELECT
    OBJECT_NAME(s.object_id) AS TableName,
    i.name AS IndexName,
    user_seeks,
    user_scans,
    user_lookups,
    user_updates,
    last_user_seek,
    last_user_scan
FROM sys.dm_db_index_usage_stats s
INNER JOIN sys.indexes i ON s.object_id = i.object_id AND s.index_id =
i.index_id

WHERE OBJECT_NAME(s.object_id) = 'Employees';
```

Анализ фрагментации индексов

```
-- Определение уровня фрагментации
SELECT
    OBJECT_NAME(ips.object_id) AS TableName,
    i.name AS IndexName,
    ips.avg_fragmentation_in_percent,
    ips.page_count
FROM sys.dm_db_index_physical_stats(DB_ID(), NULL, NULL, NULL,
'DETAILED') ips
```

```
INNER JOIN sys.indexes i ON ips.object_id = i.object_id AND ips.index_id =
i.index_id
WHERE ips.avg_fragmentation_in_percent > 10

ORDER BY ips.avg_fragmentation_in_percent DESC;
```

Обслуживание индексов

```
-- Перестроение индекса (полная реорганизация)
ALTER INDEX IX_Employees_LastName ON Employees REBUILD;

-- Реорганизация индекса (дефрагментация)
ALTER INDEX IX_Employees_LastName ON Employees REORGANIZE;

-- Обновление статистики

UPDATE STATISTICS Employees WITH FULLSCAN;
```

Автоматизация обслуживания

```
-- Динамическое перестроение индексов с высокой фрагментацией

DECLARE @TableName NVARCHAR(255);
DECLARE @IndexName NVARCHAR(255);
DECLARE @Fragmentation FLOAT;
DECLARE @SQL NVARCHAR(MAX);

DECLARE IndexCursor CURSOR FOR
SELECT
    OBJECT_NAME(ips.object_id),
    i.name,
    ips.avg_fragmentation_in_percent
FROM sys.dm_db_index_physical_stats(DB_ID(), NULL, NULL, NULL,
'SAMPLED') ips
INNER JOIN sys.indexes i ON ips.object_id = i.object_id AND ips.index_id =
i.index_id
WHERE ips.avg_fragmentation_in_percent > 30
  AND ips.page_count > 1000;

OPEN IndexCursor;
FETCH NEXT FROM IndexCursor INTO @TableName, @IndexName,
@Fragmentation;
```

```
WHILE @@FETCH_STATUS = 0
BEGIN
    IF @Fragmentation > 30
    BEGIN
        SET @SQL = 'ALTER INDEX ' + QUOTENAME(@IndexName) +
                ' ON ' + QUOTENAME(@TableName) + ' REBUILD;';
        EXEC (@SQL);
        PRINT 'Перестроен индекс: ' + @IndexName;
    END
    ELSE IF @Fragmentation > 10
    BEGIN
        SET @SQL = 'ALTER INDEX ' + QUOTENAME(@IndexName) +
                ' ON ' + QUOTENAME(@TableName) + ' REORGANIZE;';
        EXEC (@SQL);
        PRINT 'Реорганизован индекс: ' + @IndexName;
    END

    FETCH NEXT FROM IndexCursor INTO @TableName, @IndexName,
@Fragmentation;
END;

CLOSE IndexCursor;

DEALLOCATE IndexCursor;
```

Планы выполнения запросов

Анализ планов выполнения

```
-- Включение отображения плана выполнения
SET SHOWPLAN_TEXT ON;
GO
SELECT * FROM Employees WHERE LastName = 'Smith';
GO
SET SHOWPLAN_TEXT OFF;

-- Графический план выполнения (в SSMS)

-- SET STATISTICS IO ON - для статистики ввода-вывода
```

Ключевые операции доступа к данным

- Table Scan - полное сканирование таблицы (медленно);
- Index Scan - сканирование всего индекса;
- Index Seek - прямое обращение через индекс (быстро);
- Key Lookup - обращение к таблице по ключу;

Практические рекомендации

Золотые правила индексации

1. Индексируйте based on queries - создавайте индексы под конкретные запросы
2. Измеряйте производительность - тестируйте влияние индексов
3. Балансируйте количество - слишком много индексов = медленные изменения
4. Мониторьте и обслуживайте - регулярно проверяйте фрагментацию
5. Используйте покрывающие индексы для часто выполняемых запросов

Распространенные антипаттерны

1. Индексы на всех столбцах;
2. Составные индексы в неправильном порядке;
3. Отсутствие обслуживания индексов;
4. Игнорирование статистики запросов;

Контрольные вопросы

1. Какие технологии доступа к базам данных вы знаете и в чем их различия?
2. Что такое индекс и каковы его преимущества и недостатки?
3. В чем разница между кластерными и некластерными индексами?
4. Как создавать и управлять составными индексами?
5. Какие методы обслуживания индексов существуют?
6. Как анализировать эффективность индексов?

Литература

1. Дейт К. Дж. Введение в системы баз данных. - Глава 13;
2. Коннолли Т., Бегг К. Базы данных: проектирование, реализация и сопровождение. - Глава 6;

3. Microsoft SQL Server Documentation: Indexes;
4. PostgreSQL Documentation: Indexes;