Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:
«Метод ускорения запросов в базе данных»

Студент ИУ7-64Б _____ Л. Е. Тартыков
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы _____ Ю.В.Строганов
(Подпись, дата) (И.О.Фамилия)

2022г.

РЕФЕРАТ

Расчетно-пояснительная записка 46 с., 13 рис., 10 табл., 30 ист., 2 прил.

Ключевые слова: *Базы Данных*, *SQL*, *Prolog*, *Исчисление предикатов*, *Параллелизм*, *Логический вывод*.

Объектом исследования является метод ускорения запросов в базе данных.

Цель работы – нахождение способа ускорения выполнения запросов в базе данных на основе распараллеливания с использованием логического языка программирования. Для достижения поставленной цели необходимо решить следующий набор задач:

– рассмотреть подходы к представлению реляционных баз данных;

– описать методы параллелизма плана запроса и логического запроса;

– выполнить анализ существующих СУБД и реализаций Prolog, способных к распараллеливанию;

– реализовать метод ускорения выполнения запросов.

В результате выполнения работы был найден способ ускорения выполнения запросов в базе данных на основе распараллеливания с использованием логического языка программирования.

Параллельная обработка запросов позволяет сократить время отклика системы для выполняемых запросов. В сочетании с механизмами реализаций Prolog удается ускорить получение результатов в среднем в 1,5 раза.

Развитие архитектуры графических процессоров создает новые подходы в исследовании данного направления. Использование большего числа ядер по сравнению с центральным процессором позволит ускорить выполнение поиска решений для логического вывода.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

БД – Базы данных.

ЯП - язык программирования.

СУБД – Система Управления Базами Данных.

План выполнения SQL-запроса – конкретный набор операций, которые необходимо выполнить для получения результата запроса.

Планировщик SQL-запросов в реляционной СУБД – часть СУБД, которая отвечает за создание итогового плана выполнения запроса.

Терм ::= <константа> | <переменная> | <составной терм>.

ВВЕДЕНИЕ

Развитие идей реляционной алгебры Э. Кодда привело к созданию языка SQL, получившее распространение во всем мире [1]. Создавая сложные запросы, разработчикам требуется грамотно выстраивать логику их выполнения, используя различные соединения или подзапросы. Как результат, скорость разрабатывания программного продукта снижается.

В 1970-е годы популярность приобретает первая реализация Prolog, за основу которой взята математическая логика [2]. Повышение уровня лингвистической формы записи знаний, приближенной к естественному языку, привлекло ученых и исследователей в эту область.

С момента основания логическое программирование было признано парадигмой с наибольшим потенциалом автоматизированного использования параллелизма. Р. Ковальски в своих работах выделяет распараллеливание как сильную сторону автоматизации исчисления предикатов [3]. Таким образом, совмещение этих двух идей позволило бы снизить время на создание продукта, что, безусловно требует профессиональных навыков.

Цель работы – нахождение способа ускорения выполнения запросов в базе данных на основе распараллеливания с использованием логического языка программирования. Для достижения поставленной цели необходимо решить следующий набор задач:

- рассмотреть подходы к представлению реляционных баз данных;
- описать методы параллелизма плана запроса и логического запроса;
- выполнить анализ существующих СУБД и реализаций Prolog, способных к распараллеливанию;
- реализовать метод ускорения выполнения запросов.

1 Аналитический раздел

1.1 Подходы к представлению реляционных БД

Подходы к организации хранения данных в БД можно представить при помощи разных моделей [4]. Приведем некоторые из них по определению Р. Рейтера [6]:

- теоретико-модельный;
- теоретико-доказательный.

Ключевые понятия, описывающие эти подходы, представлены в таблице 1.1.

Таблица 1.1 – Подходы к организации хранения данных в БД

Теоретико-модельный	**Теоретико-доказательный**
интерпретация	гипотеза
модель	доказательство теоремы
реляционная структура	база знаний

В теоретико-модельном подходе основным компонентом представления данных является *таблица*. С математической точки зрения она представляет собой декартовое произведение $D_1, ...D_n$ доменов. Другими словами, содержит множество кортежей вида $\langle x_1, ..., x_i, ..., x_n \rangle; i = \overline{1, n}$, включающее схему отношения.

Теоретико-доказательный подход основывается на исчислении предикатов первого порядка. Использование дизъюнктов Хорна позволяет описывать отношения в виде составного терма вида $f(t_1, ..., t_n)$, где $t_1, ..., t_n$ также являются термами, описанные с использованием доменов.

Р. Ковальски [5] и Р. Рейтер [6] показали, что представление баз данных может быть рассмотрено как с теоретико-модельной, так и с доказательной точек зрения. Особенности, присущие каждому из перечисленных способов организации данных, представлены в таблице 1.2.

Каждое отношение характеризуется своим именем и списком именованных атрибутов. Такой упорядоченный набор значений является кортежем в реляционном случае. Для логического подхода такую основу составляет ис-

Таблица 1.2 – Особенности представления модели в SQL и логических ЯП

Модель	**SQL**	**Логические ЯП**
База данных	Таблица	Набор термов
Описание записи	Кортеж	Терм

пользование составного терма. Общий вид представлен в формуле (1.1).

$$R(A_1, A_2, ..., A_n), \tag{1.1}$$

где R – имя отношения, а A_i – атрибуты; $i = \overline{1, n}$. Важным свойством для предикатов является то, что они не имеют атрибутов. Единственным способом, при котором можно указывать на домен, является позиция терма A_i.

Связь этих моделей представлена на рисунке 1.1 [7].

Рисунок 1.1 – Связь моделей.

1.2 Теория множеств и базы данных

Взаимосвязь данных в БД осуществляется при помощи описания множеств – совокупности элементов, обладающих некоторым общим свойством. Используя данное понятие, можно построить более сложные и содержательные объекты.

Известно, что алгебра есть множество вида $A = \langle H, S \rangle$, где H – носитель (множество отношений), S – сигнатура (множество операций над отношениями). В реляционном случае над множествами R1 и R2 поддерживаются такие стандартные теоретико-множественные операции, как объединение ($R_1 \cup R_2$),

пересечение ($R_1 \cap R_2$), разность ($R_1 \setminus R_2$), декартово произведение ($R_1 \times R_2$). Логика первого порядка имеет свой набор операций: отрицание ($\neg$), конъюнкция ($\wedge$), дизъюнкция ($\vee$) и др, кванторы существования ($\exists$), общности ($\forall$) и т.д. Таким образом, логическая программа может быть использована для выражения практически любой операции реляционной алгебры.

1.3 Обработка SQL-запроса

Подход к выполнению SQL-запроса в СУБД представлен на рисунке 2.1.

Рисунок 1.2 – Выполнение SQL-запроса в СУБД.

Для выполнения запроса СУБД требуется в качестве входных данных составленный некоторый SQL-запрос. После пройденных этапов выполнения его обработки в результате будут получены строки, удовлетворяющие ему. Алгоритмы лексического, синтаксического, семантического анализа, планирования и оптимизации, а также система обработки правил определяется конкретной СУБД.

1.4 Структура плана запроса

Плана запроса можно представить в виде древовидной структуры, имеющей множество уровней. На нижнем находятся те узлы, на которых выполнялось сканирование таблицы для доступа к данным. При необходимости

таких операций как объединение, сортировка, агрегатные вычисления и др. соответствующие узлы добавляются над узлами сканирования. Пример такого дерева с ограниченной грамматикой SQL (содержатся основные операторы SELECT, FROM, WHERE) представлен на рисунке 1.3.

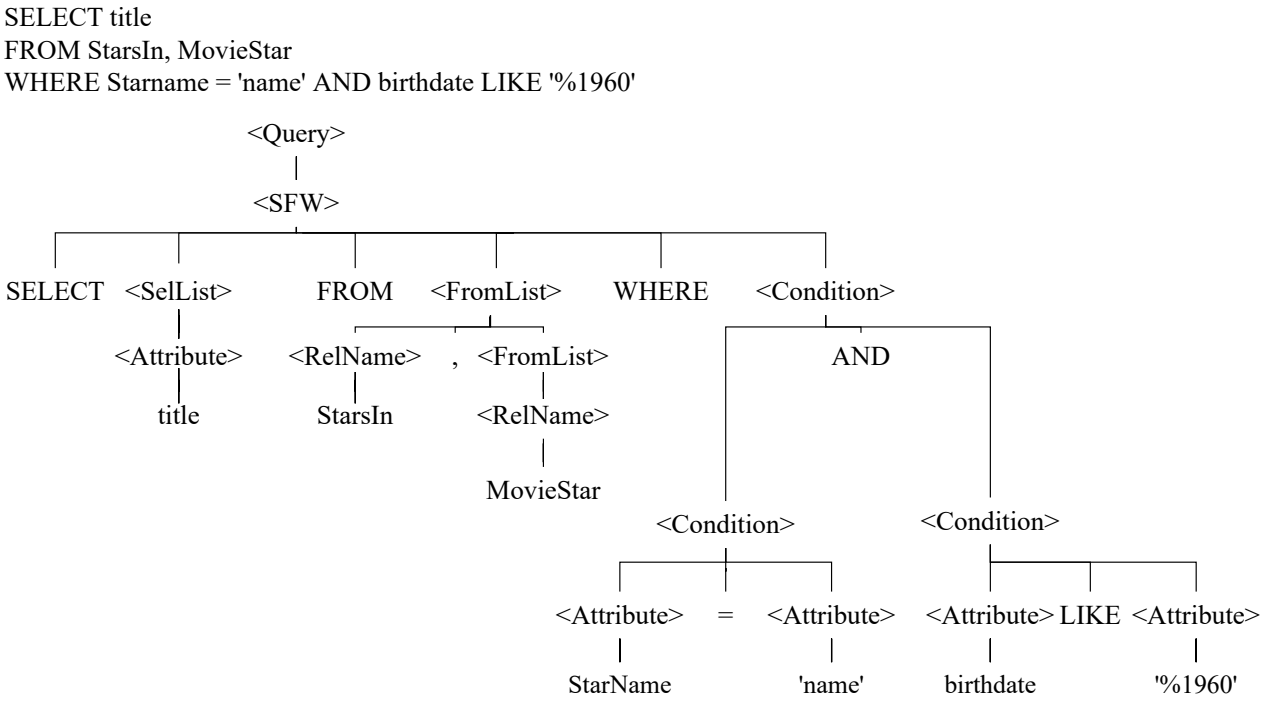

Рисунок 1.3 – Дерево плана запроса с ограниченной SQL-грамматикой.

1.5 План выполнения запроса

План выполнения запроса действует как дерево инструкций, которым должен следовать механизм выполнения запроса для получения результатов. Он показывает, как будут сканироваться таблицы; если необходимо связывание нескольких таблиц, то какой алгоритм будет выбран для объединения считанных строк. Выполнение плана на верхнем уровне представлено на рисунке 1.4.

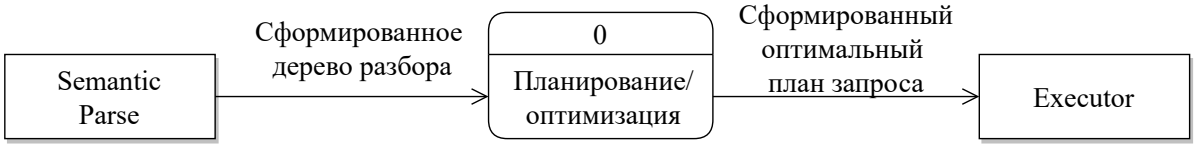

Рисунок 1.4 – Выполнение запроса (верхний уровень).

Вывод

В данном разделе были рассмотрены подходы к представлению реляционных баз данных, а также моделей в SQL и логических языках программирования. Приведено применение теории множеств к базам данных: предикаты позволят лингвистически проще описать знания, а логика предикатов может выражать почти любые операции реляционной алгебры. Описаны обработка SQL-запроса, план запроса и его структура в ограниченной грамматике SQL.

2 Конструкторский раздел

В данном разделе описываются основные шаги выполнения SQL-запроса в СУБД. Приводится алгоритм построения запроса планировщиком PostgreSQL, его стоимостная модель. Представлено описание запроса в исчислении предикатов и его сравнение с SQL версией. Излагаются подходы к распараллеливанию SQL-запросов, плана запроса и выделение независимых частей последовательности предикатов.

2.1 Основные шаги выполнения SQL-запроса

Основные шаги, которые выполняются при выполнении SQL-запроса в реляционных СУБД, представлены на рисунке 2.1 [8].

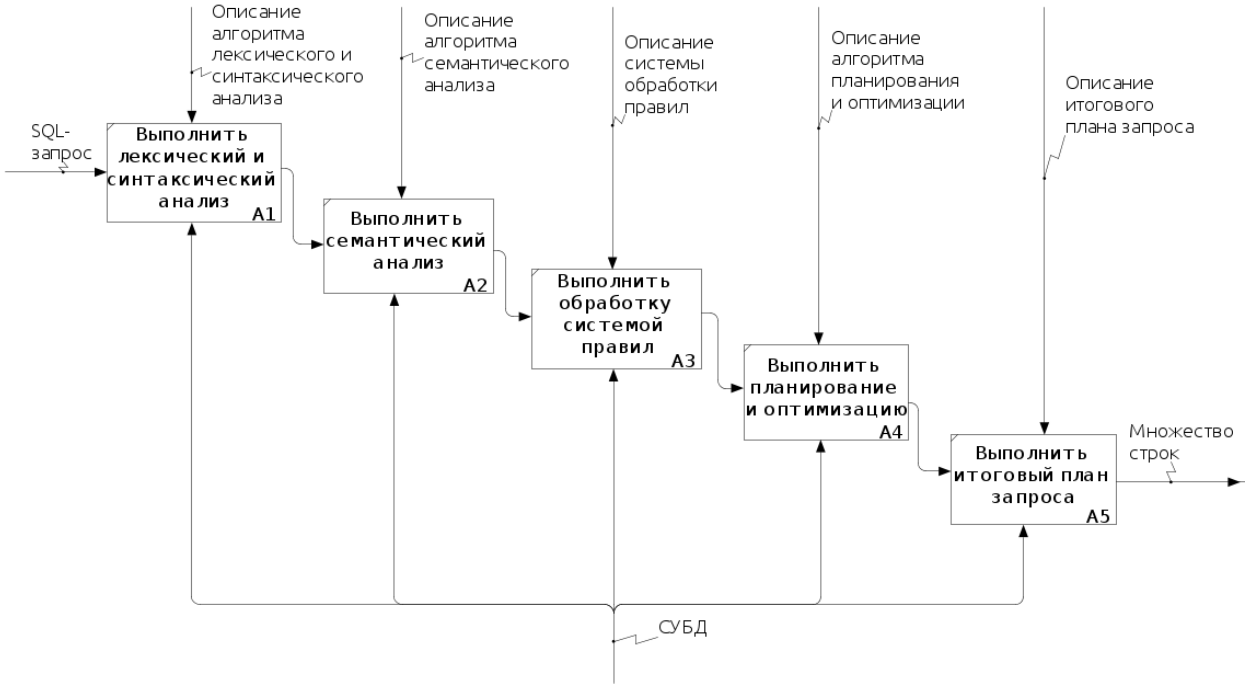

Рисунок 2.1 – Выполнение запроса в СУБД.

1. Лексический и синтаксический анализ. Входная строка пользователя обрабатывается лексическим и синтаксическим анализаторами; в результате строится дерево запроса.
2. Семантический анализ. Полученное дерево запроса дополняется различного рода метаинформацией: системными идентификаторами таблиц, типами и порядковыми номерами запрашиваемых полей, перечнем соединяемых таблиц и т.д.

3. Обработка системой правил. Выполняется поиск в системных каталогах правил, применимых к дереву запроса, и при обнаружении подходящих выполняются преобразования, которые описаны в теле найденного правила.

4. Планирование и оптимизация. На вход планировщику поступает структура с деревом запроса (рисунок 1.3). Осуществляется выбор наиболее эффективного пути выполнения этого запроса с точки зрения имеющихся оценок затрат и статической информации на момент выполнения. После выбора оптимального метода доступа к данным, конечный вариант преобразуется в полноценный план запроса и передается исполнителю.

5. Выполнение итогового плана запроса. Исполнителем осуществляется рекурсивный обход по дереву плана: *сканируются отношения*, выполняется *сортировка* и *соединения*, вычисляются *условия фильтра* и др. После выполненных этапов возвращается результирующее множество строк.

2.2 Алгоритм построения запроса в PostgreSQL

В аналитическом разделе была рассмотрена работа планировщика на верхнем уровне. Теперь же стоит заглянуть более детально в каждый из разделов и выяснить, как обрабатывается запрос, содержащий простые операторы SELECT, FROM, WHERE.

Алгоритм построения запроса в PostgreSQL представлен следующим образом: на рисунке 2.2 – более подробно описаны шаги выполнения, необходимы для модификации основной части; рисунок 2.3 отражает этапы предварительной обработки создания основного плана запроса; рисунок 2.4 представляет непосредственно базисные функции для выбора оптимального пути [9, 10].

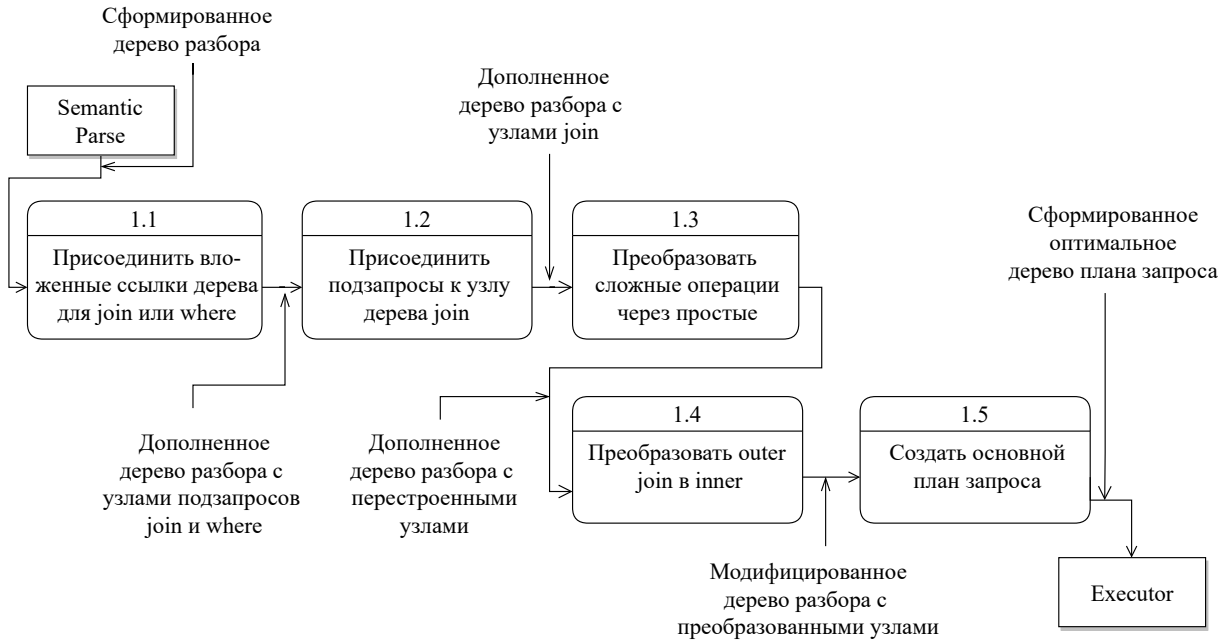

Рисунок 2.2 – Шаги выполнения планировщика и оптимизатора

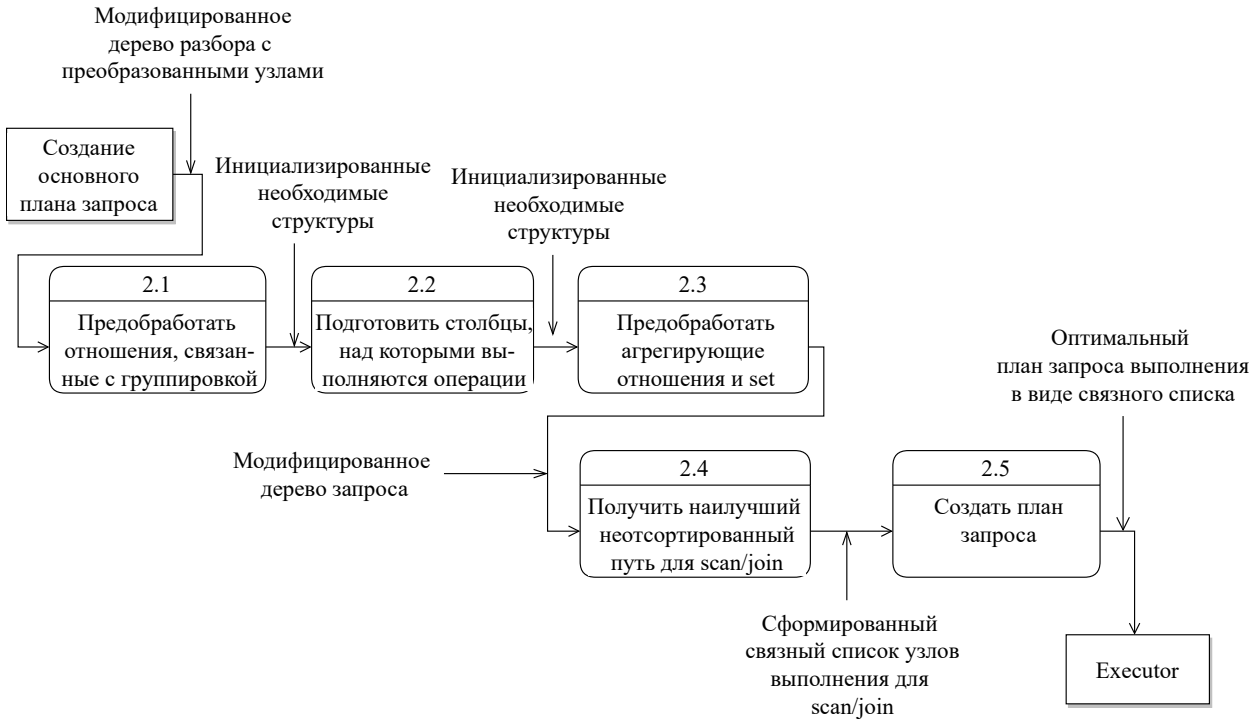

Рисунок 2.3 – Шаги выполнения предварительной обработки для создания основного плана запроса

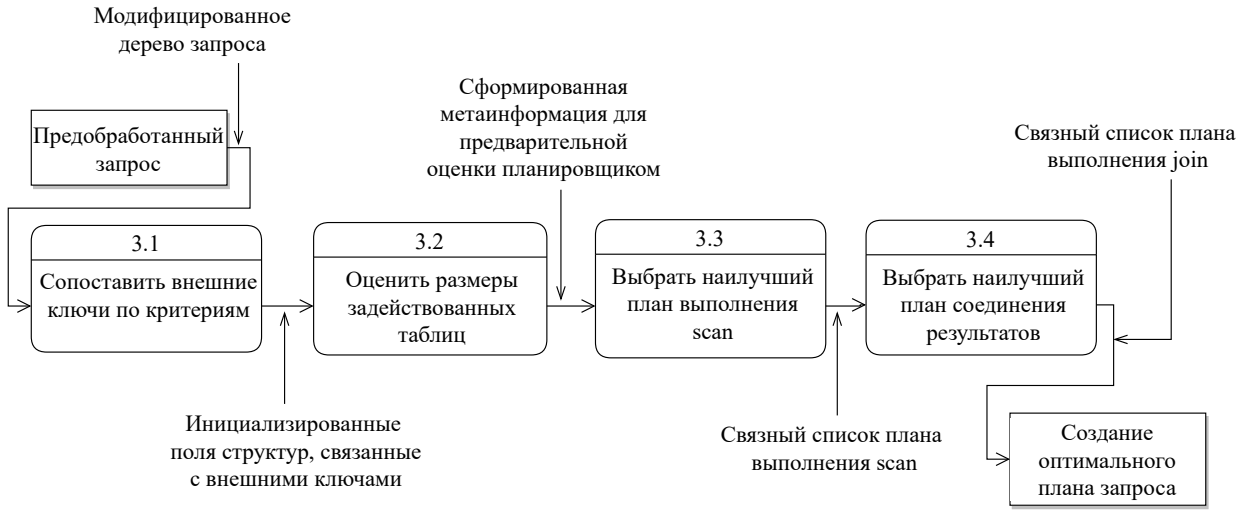

Рисунок 2.4 – Оценка и создание плана итогового запроса из оптимальных планов его узлов

В процессе выбора наилучшего пути выполнения scan (3.3) выполняются действия по сравнению стоимости из учета выполнения и статистики следующих основных способов сканирования:

- *sequential scan* – последовательное сканирование всех строк таблицы. Если был найден результат в виде одной записи вначале, то операция поиска будет продолжена;
- *index scan* – сканирование осуществляется путем использования индекса и, обычно, обхода B-дерева;
- *index only scan* – аналогично выполнению index scan за исключением того, что данные извлекают из индекса, а не из самой таблицы;
- *bitmap scan* – служит для поиска по нескольким индексам одновременно. Состоит из двух частей:
  - *bitmap index scan* – читает индекс и строит битовую карту;
  - *bitmap heap scan* – читает табличные страницы, используя построенную карту и др.

Процесс выбора наилучшего пути выполнения join (3.4) выполняет аналогичные действия, что и процесс 3.3, но связанные с объединением результатов:

- *hash join* – создается хэш-таблица на основе меньшей таблицы с ключом из join, загружая ее при этом в память целиком (если это возможно). Затем сканируется другие множества кортежей, и осуществляется

проверка этой хэш-таблицы с внешней по ключам, используемым в соединении. Предпочтительно использовать в случае, когда обе таблицы содержат более 1000 записей, и используется оператор равенства.

- *merge join* – выполняет объединение двух таблиц путем «слияния» – объединения записей одной таблицы с записями из другой; при этом лучше, чтобы данные были отсортированными по ключу; используется, когда кортежи могут быть отсортированы наилучшим способом по сравнению с другими методами;
- *nested loop join* – для каждой строки первой таблицы выполняется сравнение с каждой строкой из второй на предмет совпадения ключей; используются, когда условие сравнения не содержит оператор равенства.

2.3 Выполнение планировщиком PostgreSQL JOIN-запроса

На примере PostgreSQL работа планировщика запросов для одной таблицы выглядит следующим образом [11].

1. Выполнение предварительной обработки.
2. Оценка всевозможных путей доступа к данным. Оцениваются затраты на последовательный доступ (*seq scan*), сканирование индексов (*index scan*), *bitmap scan*; затем выполняется сортировка (*sort*) или присоединение данных (*join*).
3. Выбор кратчайшего пути по затратам.

При увеличении числа таблиц, участвующих в запросе, к основному алгоритму добавляются шаги. Процесс получения «дешевого» доступа к данным приведен на рисунке 2.5.

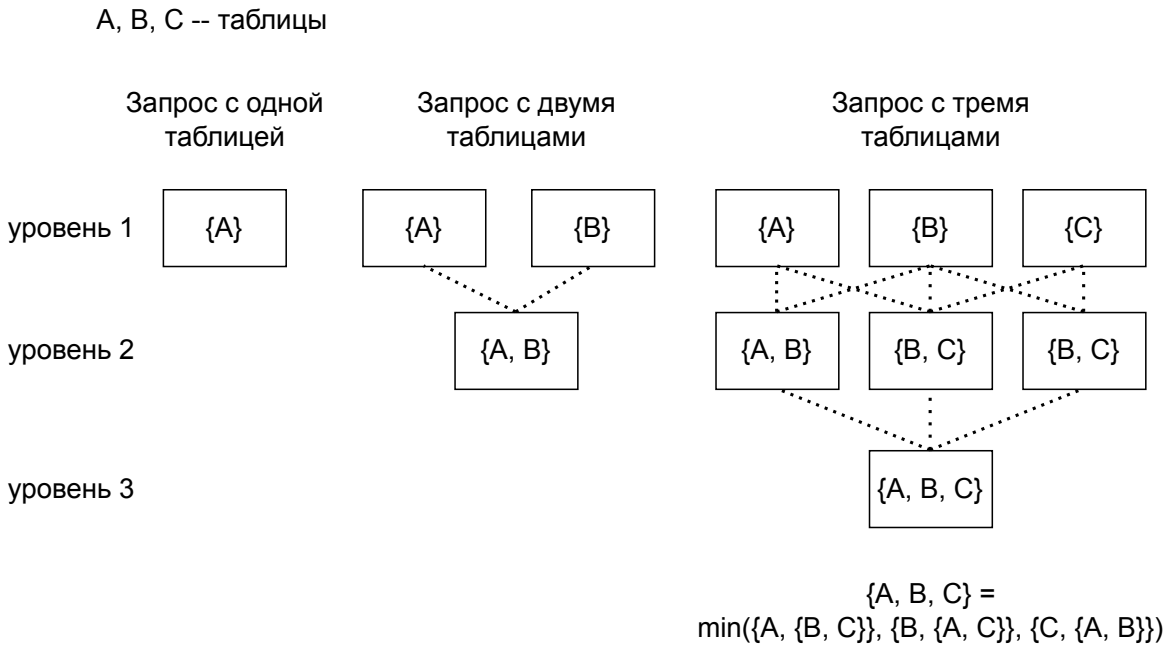

Рисунок 2.5 – Получение «дешевого» доступа к данным.

- Уровень 1. Найти кратчайший путь выполнения запроса для каждой таблицы.
- Уровень 2. Получить самый кратчайший путь для каждой комбинации, в которой которой выбирается две таблицы из всех.
- Уровень 3. Продолжить ту же обработку, пока в результате число таблиц не станет равным текущему уровню.

Для получения оптимального дерева плана запроса, планировщик должен рассмотреть комбинации всех индексов и возможности методов объединения. При увеличении числа используемых таблиц может наступить «комбинаторный взрыв». Таким образом, при количестве таблиц, большем 12, используются *генетические алгоритмы* [12].

2.4 Стоимостная модель PostgreSQL

В качестве критерия выбора единственного плана запроса из некоторого множества в движках СУБД таких, как PostgreSQL, MySQL и др., используется стоимостная модель. Само понятие «стоимость» отражает абстракцию сравнения выбора того или иного узла при построении результирующего дерева [13]. Если запрос может быть построен несколькими способами, то будет выбран тот, который имеет наименьшую стоимость. В такой модели каждая

вершина дерева плана может иметь один следующих параметров, приведенных в таблице 2.1. Все заданные коэффициенты являются относительными *seq_page_cost*.

Таблица 2.1 – Сравнение основных характеристик СУБД

Обозначение	Коэффициент	Значение	Описание
c_s	seq_page_cost	1.0	Чтение одной страницы с диска в серии последовательных чтений
c_r	random_page_cost	4.0	Чтение одной произвольной страницы с диска
c_t	cpu_tuple_cost	0.01	Обработка каждой строки при выполнении запроса
c_i	cpu_index_ tuple_cost	0.01	Обработка каждой записи индекса при выполнении запроса
c_o	cpu_operator_cost	0.0025	Обработка оператора или функции

Стоимость узла верхнего уровня включает стоимость всех его потомков. Она отражает только те факторы, которые учитывает планировщик, и не зависит от времени, которое необходимо для передачи результирующих кортежей клиенту. Общая стоимость вычисляется по формуле (2.1):

$$n_s c_s + n_r c_r + n_t c_t + n_i c_i + n_o c_o \tag{2.1}$$

2.5 Описание запроса на логическом ЯП

Для представления утверждений (по аналогии с кортежами из реляционного подхода) в логике предикатов используются кванторы. Их можно рассматривать как краткие обозначения. Например, выражение $(\forall x)\ P(x)$ обозначает «для каждого x высказывание P(x) истинно», а $(\exists x)\ P(x)$ – «существует такой x, что высказывание P(x) истинно». Это можно представить в виде формул (2.2) и (2.3):

$$(\forall x)\ (P(x)) \equiv (P(d_0) \wedge P(d_1) \wedge ... \wedge P(d_n)) \tag{2.2}$$

$$(\exists x)\ (P(x)) \equiv (P(d_0) \vee P(d_1) \vee ... \vee P(d_n)), \tag{2.3}$$

где $\{d_0, ..., d_n\}$ – домены. Формулы (2.2) и (2.3) позволяют составлять запросы к хранилищам данных на основе математического аппарата, используя исчисление предикатов первого порядка.

2.6 Логические запросы и запросы SQL

Декларативная парадигма программирования основывается на спецификации решения задачи, то есть описание проблемы и ожидаемого результата. Такой подход удобен, так как предоставление общих инструкций является более наглядным и простым методом по сравнению с формулированием цели в императивном стиле.

Данная идея заложена в основе языка SQL. Для получения результата необходимо описать такую последовательность действий, которая позволит извлечь или модифицировать данные – в данном случае это кортеж.

Для формального описания задачи логическое программирование использует логику предикатов [14]. Такой стиль позволяет на основе заданных фактов и правил вывода получить новую информацию из исходной.

Существует некоторое отличие в представлении данных в виде отношения и предикатов логических языков: в первом случае нельзя хранить какие-то вложенные структуры. Напротив, аргументы предикатов могут быть сколь угодно сложными [15].

2.7 Подходы к распараллеливанию SQL-запросов

Параллельная обработка запроса направлена на разделение одной большой задачи на небольшие, выполняющиеся в разных потоках. Благодаря такому способу сокращается время отклика системы для операций с интенсивным использованием данных, например, в хранилищах или в системах поддержки принятия решений (DSS).

Выделяют различные подходы, в которых можно выполнить распараллеливание SQL запроса [16]:

- уровень доступа – tables scan, index full scans, partitioned index range scans;
- уровень присоединения – nested loop/merge/hash join;

- DDL операции – create, drop, alter;
- DML операции – select, insert, update.

Если модуль данных не является разделяемым, то выполняются следующие шаги:

1. Разделение всех отношений, которые используются в запросе.
2. Назначение отдельной группы отношений процессору (вместе с его модулем памяти).
3. Перемещение промежуточного результирующего набора другому процессору для оценки результата.

Стоит отметить, что данный метод применим только для операции select.

2.8 Подходы к распараллеливанию плана запроса

Немаловажную роль играют сами узлы построенного планировщиком дерева плана запроса. В основе идеи ускорения лежит обработка одновременно тех частей, которые до момента своего завершения не позволяют перейти на другой узел выполнения (в случае последовательной реализации) [17]. Графически данный способ представлен на рисунке 2.6.

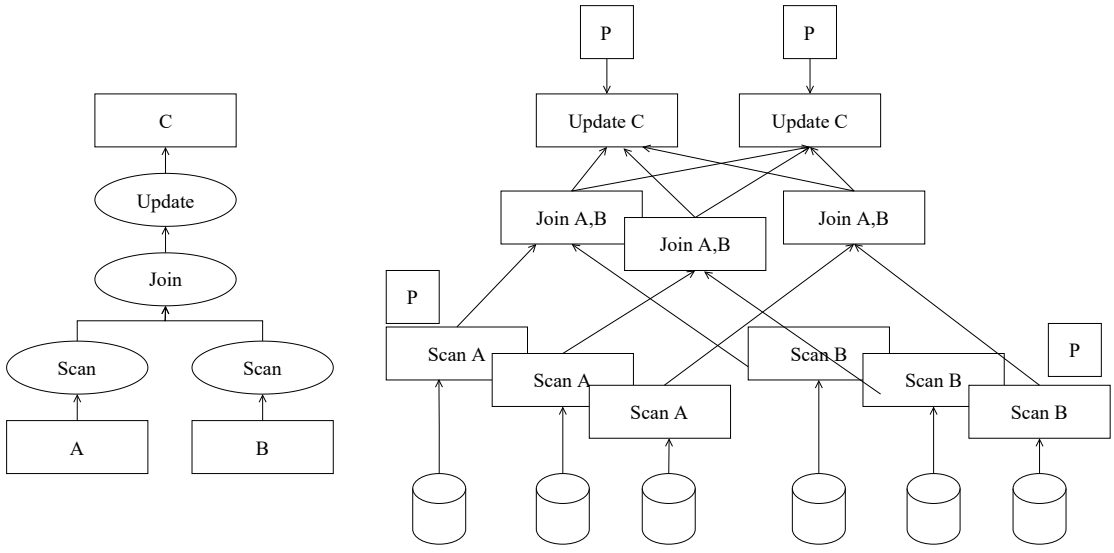

Рисунок 2.6 – Узлы планировщика как идея для распараллеливания.

2.9 Выделение независимых частей логического запроса

Рассмотрим запрос $(\forall v_i)(\forall p)(P \wedge B_1 \wedge ... \wedge B_n \rightarrow A)$, где $P, B_1, ..., B_n$ – независимые предикаты, а v_i, p – переменные, относящиеся к ним. В случае логических языков программирования отдельное внимание уделяется обработке таких предикатов. Пусть для предиката P время поиска ответа больше по сравнению с последующими из данной конъюнкции. Тогда системе приходится ждать, пока найдется такое решение (или его может не быть), чтобы перейти к дальнейшим шагам выполнения. Возникает вопрос о независимой и одновременной обработке таких предикатов.

Для решения данной проблемы выделяют следующие подходы [18].

1. AND-параллелизм (независимый).
2. OR-параллелизм (независимый).

Рассмотрим более подробно каждый из них.

- В основе идеи независимого AND-параллелизма лежит распараллеливание конъюнкции подцелей, которые не имеют одни и те же переменные. Такие термы не будут влиять на выполнение друг друга, устраняя необходимость введения какой-либо формы синхронизации во время параллельного выполнения. Данный способ проиллюстрирован на рисунке 2.7.

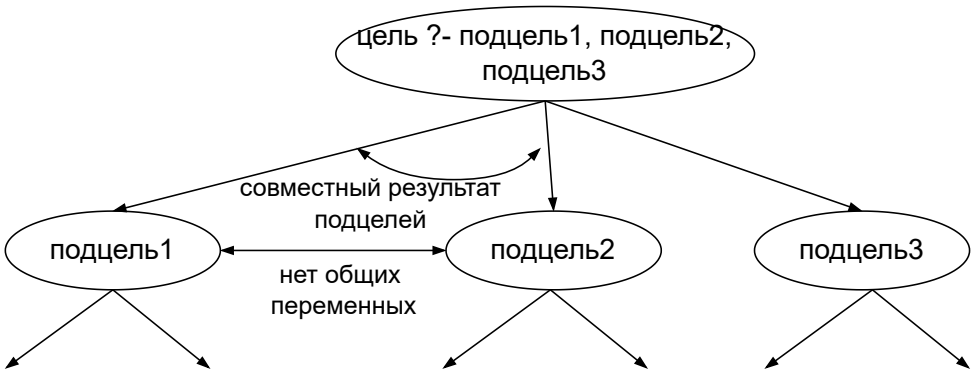

Рисунок 2.7 – AND-дерево.

- Независимый OR-параллелизм основан на параллельном выполнении дизъюнкции предикатов, не имеющих общих переменных. Результат

будет получен из первого выполнившегося терма. Пример OR-дерева приведен на рисунке 2.8.

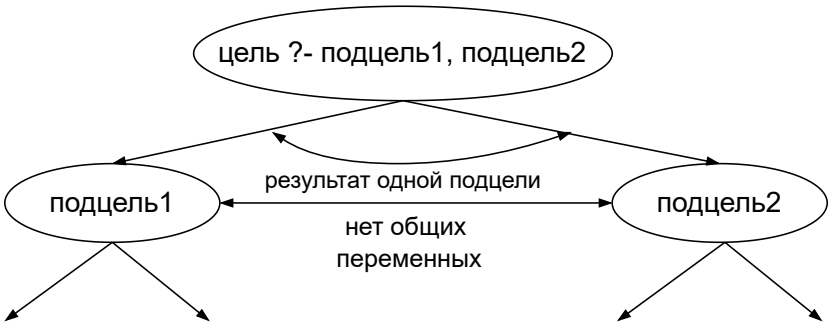

Рисунок 2.8 – OR-дерево.

Вывод

В данном разделе были описаны основные шаги выполнения SQL-запроса в СУБД. Приведен алгоритм построения запроса планировщиком PostgreSQL, его стоимостная модель. Представлено описание запроса в исчислении предикатов и его сравнение с SQL версией. Были изложены подходы к распараллеливанию SQL-запросов, плана запроса и выделение независимых частей последовательности предикатов.

3 Технологический раздел

3.1 Запросы реляционной алгебры и их эквивалентное описание в логике предикатов

На листинге 3.1 приводятся примеры описания простых операций выборки, сравнения, используя реляционную алгебру и логику предикатов [19].

Листинг 3.1 – Сопоставление операций

Реляционная алгебра	Логика предикатов
1. Проекция	
$\mathrm{S} = \pi_{parent\_name}(Parent)$	$\mathrm{S(Y)} \leftarrow \mathrm{parent(X,\ Y)}$
2. Выборка	
$\mathrm{millionare} = \sigma_{amount}(Income)$	$\mathrm{millionare} \leftarrow \mathrm{income(X,\ Y)}$
3. Фильтр по признаку	
$\mathrm{millionare} = \sigma_{amount>10000}(Income)$	$\mathrm{millionare} \leftarrow \mathrm{income(X,\ Y)}, \mathrm{Y} > 10000$
4. Объединение	
$R_1(A_1, A_2) \cap R_2(A_1, A_2)$	$P(d_1, d_2) \wedge P(d_3, d_4)$
5. Пересечение	
$R_1(A_1, A_2) \cup R_2(A_1, A_2)$	$P(d_1, d_2) \vee P(d_3, d_4)$

3.2 Сравнение СУБД

Ограничения к доступу коммерческими компаниями своих продуктов на фоне различных конфликтов приводят к решению импортозамещения программных решений. Возникает необходимость использования открытых ПО. Сравнение основных характеристик СУБД приводится в таблице 3.1 [20].

PostgreSQL [21] является стабильной СУБД [22], имеет хорошее структурирование, поэтому в качестве основной выбрана она.

Таблица 3.1 – Сравнение основных характеристик СУБД

СУБД	Open source	ACID	RDBMS	Cloud-only	OLTP	In-memory	SQL
PostgreSQL	+	+	+	-	+	-	+
Oracle	-	+	+	-	+	-	+
MySQL	+	+	+	-	+	-	+
MariaDB	+	+	+	-	+	-	+
MongoDB	+	+	-	-	+	-	-
SQLite	+	+	+	-	+	-	+
Cassandra	+	-	-	-	+	-	-
Redis	+	-	-	-	+	+	-

3.3 Сравнение учебных БД

«Adventure Works Cycles» – фиктивная компания, разработанная Microsoft для моделирования бизнес-процесса [23]. Данная фирма «является» крупной и многонациональной, которая производит и продает металлические и композитные велосипеды на коммерческих рынках Северной Америки, Европы, Азии. По своей структуре база данных «AdventureWorks» является сложной: состоит из множества схем отношений, соединенных друг с другом различными связями. Диаграмма базы данных этой компании приведена в приложении А на рисунке А.1.

База данных *«Northwind»* содержит данные о продажах фиктивной компании под названием «Northwind Traders», которая импортирует и экспортирует специальные продукты питания со всего мира [24]. Диаграмма БД этой компании приведена в приложении А на рисунке А.2.

Модель данных *«Chinook»* представляет собой хранилище цифровых медиа, включая таблицы для исполнителей, альбомов, аудиодорожек, счетов-фактур и клиентов [25]. Диаграмма БД этой компании приведена в приложении А на рисунке А.3.

В тестируемой базе данных в нескольких таблицах должно присутствовать число записей, превышающее 10^6 штук. Такому требованию удовлетворяет «AdventureWorks», поэтому в качестве учебной была выбрана именно она.

3.4 Выбор реализаций Prolog, способных к распараллеливанию

Существует множество версий параллельного Prolog. Необходима такая реализация, которая способна отвечать следующим критериям:

- ПО должно быть открытым (критерий 1);
- время загрузки программы не должно превышать 10 минут (критерий 2);
- программа не должна занимать оперативной памяти больше 5Гб (критерий 3);
- программа должна возвращать результат с числом столбцов, большем чем 1 (критерий 4);
- должно быть загружено число записей, превышающее 10^6 (критерий 5).

В таблице 3.2 представлены реализации Prolog и их удовлетворение критериям.

Таблица 3.2 – Сравнение основных реализаций Prolog

Реализация	Крит. 1	Крит. 2	Крит. 3	Крит. 4	Крит. 5
Datalog	+	-	+	+	+
Eclipse	+	+	-	+	-
Visual Prolog	-	?	?	?	?
CIAO	+	-	+	+	+
SWI-Prolog	+	+	+	+	+
Parlog	+	+	+	-	-
SICStus	-	?	?	?	?

Знак «?» означает, что в свободном доступе проверить удовлетворение критерию невозможно.

На основе вышеперечисленных требований была выбрана реализация SWI-Prolog [26], которая удовлетворяет всем критериям.

3.5 Выбор средств реализации

Для запуска приложения используется принцип контейнеризации на примере Docker [27]. Он позволяет упаковывать все имеющиеся зависимости, их версии и осуществлять запуск в изолированной среде. Таким образом, можно работать с несколькими контейнерами на одном хосте, легко ими делиться для других пользователей на разных компьютерах. Инструкция по развертыванию ПО приведена в приложении Б на листинге Б.1.

3.6 Детали реализации

На листинге 3.2 приведен выполняемый простой SQL-запрос (содержит операторы SELECT, FROM, WHERE). Версия, написанная через конъюнкцию предикатов, представлен на листинге 3.3.

Листинг 3.2 – Простой SQL-запрос

```
select *
from comp_one
where surname = 'Гусева' and age > 40;
```

Листинг 3.3 – Простой запрос в версии SWI-Prolog

```
bigger_age(Id, Surname, Age, Place, MS, MA) :-
    comp_one(Id, Surname, Age, Place),
    Surname = MS, Age > MA.

findall((Id, Surname, Age, Place), bigger_age(Id, Surname, Age,
   Place, 'Гусева', 40), Ids).
```

На листинге 3.4 содержится запрос, имеющий операторы SELECT, FROM, WHERE, а также обращение к другой таблице при помощи JOIN по внешнему ключу, записанный на PostgreSQL. Запрос через конъюнкцию предикатов представлен на листинге 3.5.

Листинг 3.4 – Запрос с оператором JOIN (PostgreSQL)

```
select c.id, c.age, a.name, a.kind
from comp_one as c join animals as a on c.id = a.id_host
where surname = 'Гусева' and c.age > 40;
```

Листинг 3.5 – Запрос с оператором JOIN (SWI-Prolog)

```
find_an(SW, AgeW, Id, Age, Name, Kind) :-
    comp_one(Id, Surname, Age, _, _, _, _, _), Surname = SW,
    Age > AgeW,
    animals(_, Idh, Name, Kind, _, _, _, _), Id = Idh).

findall((Id, Age, Name, Kind),
        find_an('Гусева', 40, Id, Age, Name, Kind), IdS).
```

На листинге 3.6 представлено использование потоков для версии PostgreSQL.

Листинг 3.6 – Запрос с оператором SELECT с использованием 5 потоков (PostgreSQL)

```
set max_parallel_workers_per_gather=16;

explain analyze select *
from comp_one
where surname = 'Гусева' and age > 40;
```

Запрос, написанный в версии Prolog с примером в виде пяти потоков, представлен на листингах 3.7 и 3.8.

Листинг 3.7 – Запрос с оператором SELECT с использованием 5 потоков (Prolog). Часть 1.

```
bigger_age(Id, Surname, Age, Cabinet, Post, WorkD, MS, MA):-
        comp_one(Id, Surname, Age, Cabinet, Post, WorkD),
        Surname = MS, Age > MA.
sel1(NSur, NAge) :-
        findall((Id, Sur, Age, Cab, Pst, WD), (bigger_age(Id, Sur,
            Age, Cab, Pst, WD, NSur, NAge), Id < 2000000), _).
sel2(NSur, NAge) :-
        findall((Id, Sur, Age, Cab, Pst, WD), (bigger_age(Id, Sur,
            Age, Cab, Pst, WD, NSur, NAge), Id >= 2000000, Id <
            4000000), _).
sel3(NSur, NAge) :-
        findall((Id, Sur, Age, Cab, Pst, WD), (bigger_age(Id, Sur,
            Age, Cab, Pst, WD, NSur, NAge), Id >= 4000000, Id <
            6000000), _).
```

Листинг 3.8 – Запрос с оператором SELECT с использованием 5 потоков (Prolog). Часть 2.

```
sel4(NSur, NAge) :-
        findall((Id, Sur, Age, Cab, Pst, WD), (bigger_age(Id, Sur,
           Age, Cab, Pst, WD, NSur, NAge), Id >= 6000000, Id <
           8000000), _).
sel5(NSur, NAge) :-
        findall((Id, Sur, Age, Cab, Pst, WD), (bigger_age(Id, Sur,
           Age, Cab, Pst, WD, NSur, NAge), Id >= 8000000), _).
testTime(NSur, NAge) :-
        statistics(cputime, OldCpu),
        get_time(OldWall),
        thread_create(sel1(NSur, NAge), Sel1Thread),
        thread_create(sel2(NSur, NAge), Sel2Thread),
        thread_create(sel3(NSur, NAge), Sel3Thread),
        thread_create(sel4(NSur, NAge), Sel4Thread),
        thread_create(sel5(NSur, NAge), Sel5Thread),
        thread_join(Sel1Thread),
        thread_join(Sel2Thread),
        thread_join(Sel3Thread),
        thread_join(Sel4Thread),
        thread_join(Sel5Thread),
        get_time(NewWall),
        statistics(cputime, NewCpu),
        UsedWall is NewWall - OldWall,
        UsedCpu  is NewCpu  - OldCpu,
        assertz( db_bench(BenchMark, stat{  cpu_time: UsedCpu,
                                  wall_time: UsedWall })),
        print_message(information, bench(UsedCpu, UsedWall)).
```

3.7 Очистка системного кэша

На листинге 3.9 приведен скрипт выполнения очистки системного кэша, что не позволяет обратиться к имеющемуся результату выполнения запроса.

Листинг 3.9 – Очистка кэша

```
service postgresql stop;
sync;
echo 3 > /proc/sys/vm/drop_caches;
service postgresql start
```

Вывод

В данном разделе было изложено сопоставление запросов в реляционной алгебре и их эквивалентное описание в логике предикатов. Также был обоснован выбор СУБД PostgreSQL, БД «AdventureWorks», реализации SWI-Prolog, поддерживающей многопоточность. Приводится выбор средств разработки, а также деталей реализации.

4 Экспериментальный раздел

4.1 Технические характеристики

Технические характеристики устройства, на котором проводились эксперименты:

- операционная система: Ubuntu 22.04 [28];
- память: 8Гб;
- размер swap-файла: 2Гб;
- процессор: Intel(R) Core(TM) i5-8265U CPU @ 1.60ГГц 1.80ГГц.

Во время проведения тестирования устройство было подключено к блоку питания и не нагружено никакими приложениями, кроме встроенных приложений окружения и самим окружением. Режим электропитания был включен на «Производительный».

4.2 SELECT-запрос

Целью эксперимента является определение зависимости выполнения запроса от числа столбцов для постоянного числа записей таблицы в базе данных (аналогично для количества правил базы знаний).

В таблице 4.1 приведены экспериментально полученные значения временных характеристик выполнения SELECT-запросов.

На рисунке 4.1 приведена зависимость выполнения простого SELECT-запроса от числа столбцов для постоянного числа записей.

Таблица 4.1 – Таблица времени выполнения простого SELECT-запроса (в секундах)

Число столбцов	Число записей	Програмное обеспечение			
		SWI Prolog	YAP Prolog	PostgreSQL (без кэша)	PostgreSQL (с кэшем)
4	10^7	0.659228	0.549383	1.339074	0.807485
5		0.711844	0.679519	1.227935	0.818097
6		0.770522	0.771545	1.193835	0.825795
7		0.823258	0.841355	1.750495	0.832614
8		0.886037	0.94406	1.366828	0.836362
9		0.938527	1.066499	2.109747	0.851722
10		0.974033	1.136439	2.058274	0.850416
11		1.04980	1.248144	2.307539	0.852188
12		1.37296	1.411219	1.607229	0.884879
13		1.64924	1.659166	1.873321	0.884116

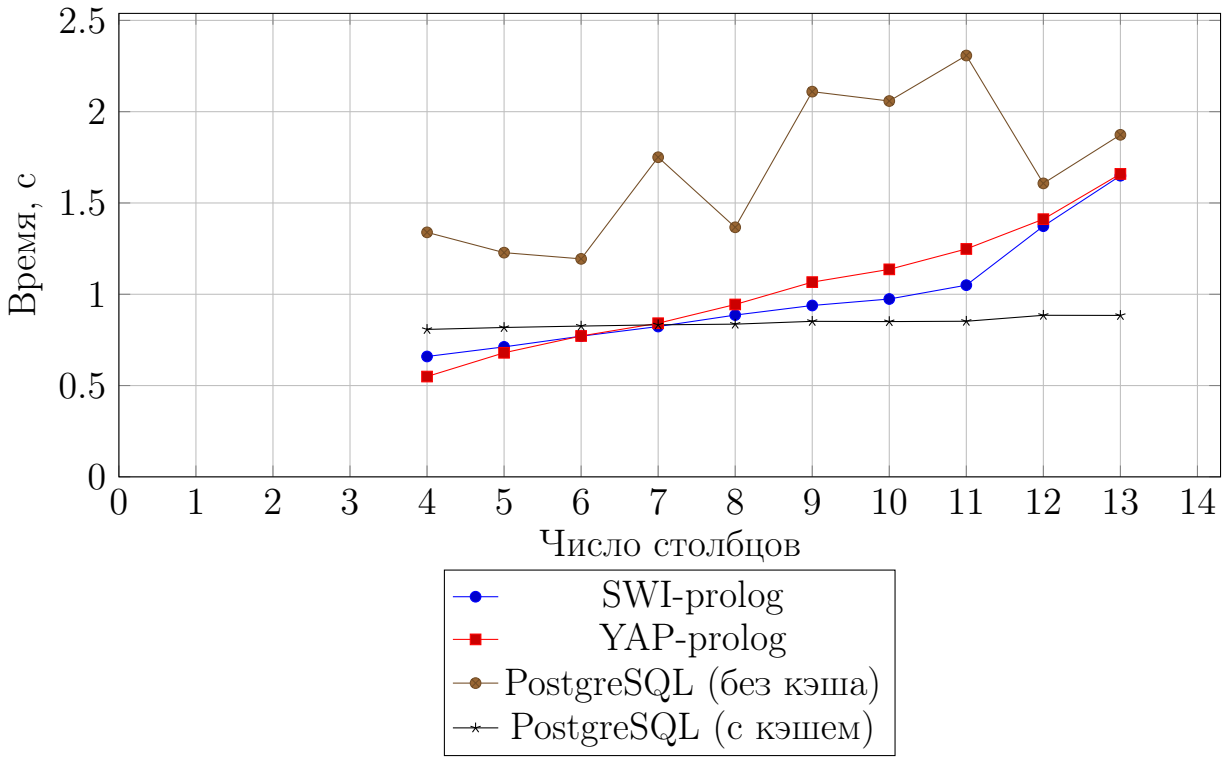

Рисунок 4.1 – Зависимость выполнения простого SELECT-запроса от числа столбцов для постоянного числа записей

4.3 JOIN-запрос

Целью эксперимента является определение зависимости выполнения запроса с обращением к другой таблице от числа столбцов для постоянного

числа записей (аналогично для количества правил базы знаний).

В таблице 4.2 приведены экспериментально полученные значения временных характеристик выполнения запросов.

Таблица 4.2 – Таблица времени выполнения JOIN-запроса (в секундах)

Число столбцов	Число записей	Программное обеспечение		
		SWI Prolog	YAP Prolog	PostgreSQL (без потоков)
4	10^5	375.436966	253.256186	0.031547
5		387.485804	367.456964	0.033847
6		438.366495	427.122528	0.038549
7		468.279248	478.549645	0.043654
8		504.606969	520.154154	0.048515
9		534.500522	561.591546	0.052371
10		554.721574	578.218155	0.055763
11		597.871645	603.125187	0.059163
12		780.914253	824.519815	0.063312
13		939.258741	1031.98327	0.069855

На рисунке 4.2 приведена зависимость выполнения JOIN-запроса от числа столбцов для постоянного числа записей.

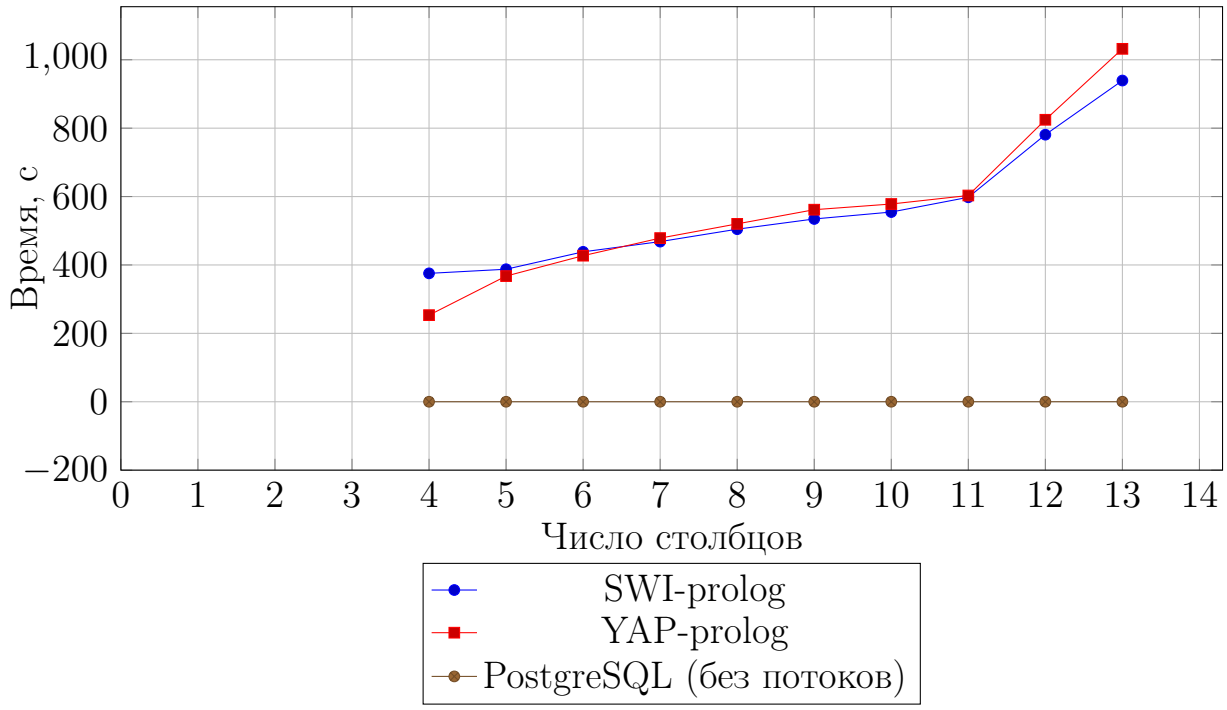

Рисунок 4.2 – Зависимость выполнения JOIN-запроса от числа столбцов для постоянного числа записей

4.4 SELECT-запрос с потоками

Целью эксперимента является определение зависимости выполнения запроса от числа столбцов таблицы базы данных с постоянным числом записей (аналогично для количества правил базы знаний) с использованием распараллеливания.

В таблице 4.3 приведены экспериментально полученные значения временных характеристик выполнения запросов. В SWI-Prolog использовалось число потоков, равное логическому числу ядер машины, на которой проводился эксперимент (8). Для версии PostgreSQL был установлен параметр максимального числа потоков, равным 16. Планировщиком был выбрано количество workers в виде 5 штук, что было выведено в плане запроса при помощи EXPLAIN ANALYZE [29].

Таблица 4.3 – Таблица времени выполнения простого SELECT-запроса с использованием потоков (в секундах)

Число столбцов	Число записей	Програмное обеспечение		
		SWI Prolog	PostgreSQL (без кэша)	PostgreSQL (с кэшем)
4	10^7	1.353939	0.989799	0.505146
5		1.481703	0.730035	0.509551
6		1.594352	0.768276	0.505604
7		1.738103	1.042243	0.515283
8		1.850044	0.833488	0.514180
9		1.990824	0.981088	0.520004
10		2.173378	1.190157	0.518509
11		2.249650	1.250775	0.522520
12		2.397863	1.251425	0.537440
13		2.560862	1.065807	0.539438

На рисунке 4.3 приведена зависимость выполнения простого SELECT-запроса с использованием потоков.

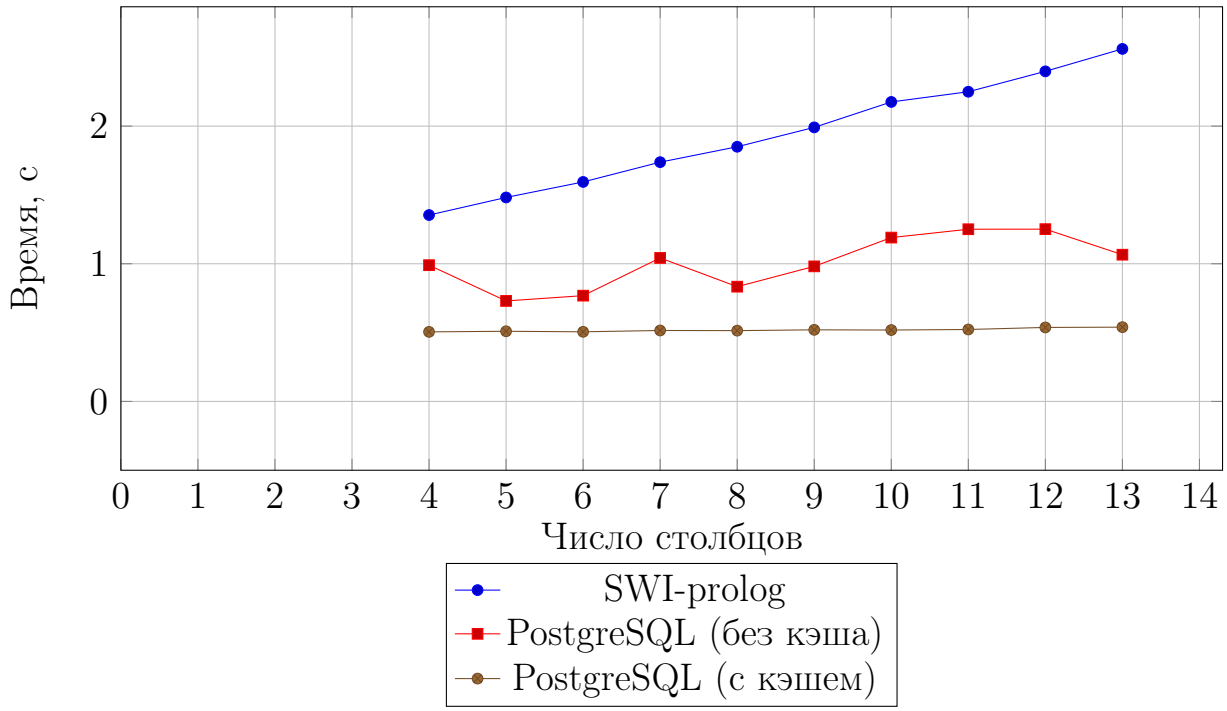

Рисунок 4.3 – Зависимость времени выполнения простого SELECT-запроса с использованием потоков

4.5 Потоки SWI-Prolog

Целью эксперимента является определение зависимости выполнения запроса от числа потоков в реализации SWI-Prolog для постоянного числа записей. Результаты выполнения измерений представлены в таблице 4.4.

Таблица 4.4 – Таблица времени выполнения простого SELECT-запроса с использованием потоков (в секундах)

Число столбцов	Число записей	Число потоков	SWI Prolog
4	10^7	2	0.592607
		3	0.673686
		4	0.736398
		5	1.012103
		6	1.099502
		7	1.230243
		8	1.353939

Продолжение на следующей странице

Таблица 4.4 – продолжение

Число столбцов	Число записей	Число потоков	SWI Prolog
5	10^7	2	0.645411
		3	0.733633
		4	0.803135
		5	1.092881
		6	1.185542
		7	1.393922
		8	1.481703
9	10^7	2	0.887928
		3	0.969058
		4	1.090071
		5	1.476527
		6	1.573091
		7	1.802725
		8	1.990824
13	10^7	2	1.375427
		3	1.224211
		4	1.524814
		5	1.863615
		6	2.01534
		7	2.275334
		8	2.560862
Конец таблицы			

В таблице 4.5 приведены совмещенные результаты, полученные в таблицах 4.3 и 4.4.

Таблица 4.5 – Объединение результатов простого SELECT-запроса с использованием потоков (в секундах)

Число столбцов	Число записей	SWI Prolog(2)	SWI Prolog(3)	SWI Prolog(4)	SWI Prolog(5)	Postgre SQL(5)
4	10^7	0.592607	0.673686	0.736398	1.012103	0.989799
5		0.645411	0.733633	0.803135	1.092881	0.730035
9		0.887928	0.969058	1.090071	1.476527	0.981088
13		1.375427	1.224211	1.524814	1.863615	1.065807

На рисунке 4.4 приведена зависимость выполнения простого SELECT-запроса с использованием потоков.

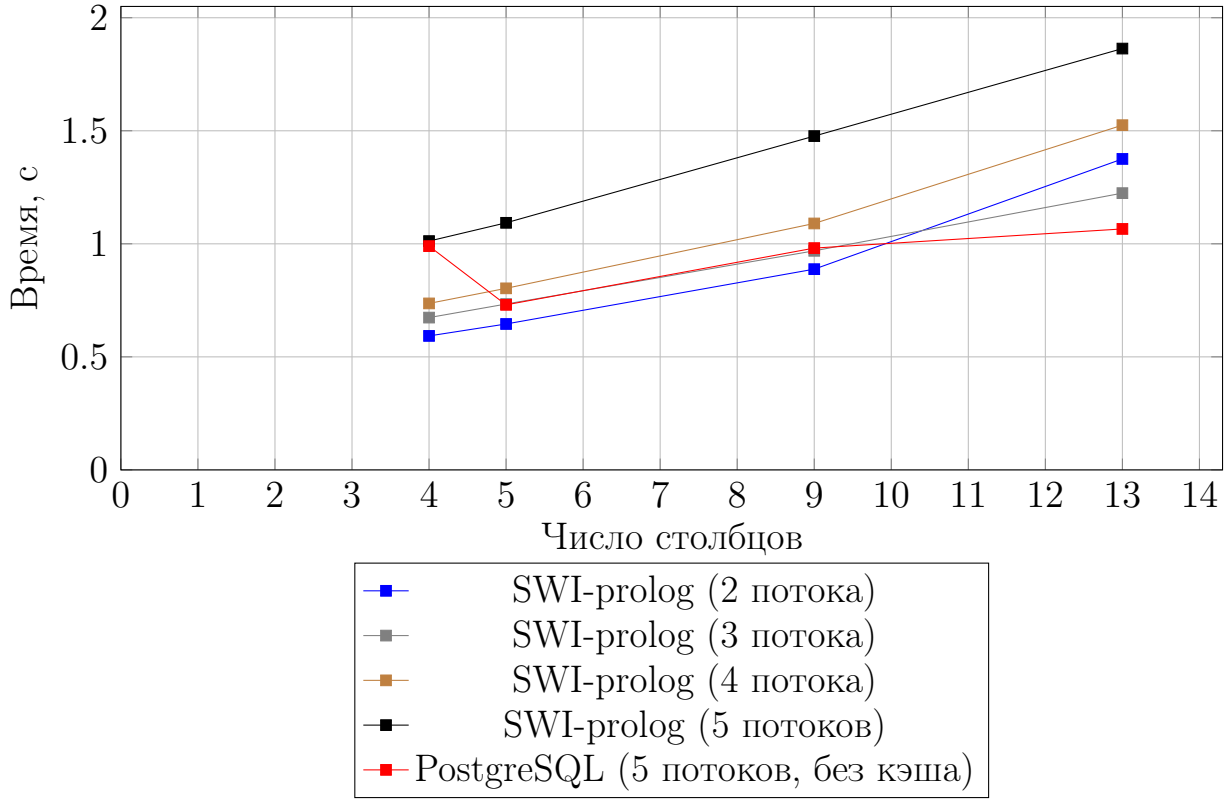

Рисунок 4.4 – Зависимость времени выполнения простого SELECT-запроса с использованием потоков

Вывод

Логический язык программирования рекомендуется как инструмент выполнения простых SELECT-запросов в таблице. Использование такого подхода будет полезным, если не требуется кэширование при вычислениях. Отличительные показатели Prolog-реализаций демонстрируются для малого чис-

ла термов (4-7). Однако, при обработке больше 8 столбцов требуется больше времени для получения результата, чем исполнителю PostgreSQL с сохранением в кэш-памяти. Стоит отметить, что YAP-Prolog начинает проседать по производительности по сравнению с SWI по мере увеличения количества атрибутов.

Обращение к данным из другой таблицы требует больше проверок, чем выполнение простого запроса. Таким образом, Prolog-версии программного обеспечения выдают временные показатели в 10000 раз выше, чем PostgreSQL.

Распараллеливание запросов механизмами логического языка программирования показало, что использование параллелизма в этой парадигме отличается от императивного стиля. Не следует работать с больше, чем двумя-тремя потоками, что приводит к увеличению нагрузки выполнения запроса.

ЗАКЛЮЧЕНИЕ

Задача лингвистического упрощения записи программ, приближенного к естественному языку, приводит ученых и исследователей к созданию новых средств реализаций на основе математического аппарата. Подтверждением этого является интерес и развитие идей автоматизация логического вывода.

Параллельная обработка запросов позволяет сократить время отклика системы для выполняемых запросов. В сочетании с механизмами реализаций Prolog удается ускорить получение результатов в среднем в 1,5 раза.

Цель работы курсовой работы достигнута – найден способ ускорения выполнения запросов в базе данных на основе распараллеливания с использованием логического языка программирования. В процессе достижения поставленной цели были решены следующие задачи:

- рассмотрены подходы к представлению реляционных баз данных;
- описаны методы параллелизма плана запроса и логического запроса;
- выполнен анализ существующих СУБД, учебных БД и реализаций Prolog, способных к распараллеливанию; обоснован их выбор;
- реализован метод ускорения выполнения запросов.

Развитие архитектуры графических процессоров создает новые подходы в исследовании данного направления. Использование большего числа ядер по сравнению с центральным процессором позволит ускорить выполнения поиска решений для логического вывода. Представление результатов эксперимента по производительности выполнения реализации GPU-Datalog [30] перспективное направление исследования в области логики.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Chamberlin D. D. Early history of SQL //IEEE Annals of the History of Computing. – 2012. – Т. 34. – №. 4. – С. 78-82.

2. Koerner P. et al. Fifty Years of Prolog and Beyond //Theory and Practice of Logic Programming. – 2022. – С. 1-83.

3. Kowalski R. Logic for Problem Solving. Elseiver North Holland //Inc. – 1979.

4. Gallaire H., Minker J., Nicolas J. M. Logic and databases: A deductive approach //Readings in artificial intelligence and databases. – Morgan Kaufmann, 1989. – С. 231-247.

5. Kowalski R. Logic as a database language //on Proc. of the third British national conference on databases (BNCOD3). – 1984. – С. 103-132.

6. Reiter R. On Conceptual Modelling, chapter Towards a Logical Reconstruction of Relational Database Theory //On Conceptual Modelling. – 1984. – С. 191-233.

7. Logica [Электронный ресурс]. URL: https://opensource.googleblog.com /2021/04/logica-organizing-your-data-queries.html (дата обращения: 16.05.2022).

8. Пантилимонов М. В., Бучацкий Р. А., Жуйков Р. А. Кэширование машинного кода в динамическом компиляторе SQL-запросов для СУБД PostgreSQL //Труды Института системного программирования РАН. – 2020. – Т. 32. – №. 1. – С. 205-220.

9. PostgreSQL Source Code [Электронный ресурс]. URL: https://doxygen.postgresql.org/postgres_8c.html#a7908e75bd9f9494fdb 8c4b47f01a9de9 (дата обращения: 11.04.2022).

10. How Postgres Chooses Which Index To Use For A Query [Электронный ресурс]. URL: https://pganalyze.com/blog/how-postgres-chooses-index (дата обращения: 22.04.2022).

11. The Internals of PostgreSQL : Chapter 3 Query Processing [Электронный ресурс]. URL: https://www.interdb.jp/pg/pgsql03.html (дата обращения: 13.04.2022).

12. PostgreSQL: Documentation: 14: 60.2. Genetic Algorithms [Электронный ресурс]. URL: https://www.postgresql.org/docs/current/geqo-intro2.html (дата обращения: 13.04.2022).

13. Postgres Pro Standard. Использование EXPLAIN [Электронный ресурс]. URL: https://postgrespro.ru/docs/postgrespro/9.5/using-explain?lang=ru (дата обращения: 02.05.2022)

14. Чистяков М. Ю. Логическое программирование как одна из парадигм программирования //52-яНАУЧНАЯ КОНФЕРЕНЦИЯ. – 2016. – С. 146.

15. Maier F. et al. PROLOG/RDBMS integration in the NED intelligent information system //Lecture Notes in Computer Science. – 2002. – С. 528-528.

16. Parallel Execution [Электронный ресурс]. URL: https://www.sdcc.bnl.gov/phobos/Detectors/Computing/Orant/doc/database.804/a58227/ch_paral.htm (дата обращения: 04.06.2022).

17. Deepak S. et al. Query processing and optimization of parallel database system in multi processor environments //2012 Sixth Asia Modelling Symposium. – IEEE, 2012. – С. 191-194.

18. Gupta G., Costa V. S. Cuts and side-effects in and-or parallel prolog //The Journal of logic programming. – 1996. – Т. 27. – №. 1. – С. 45-71.

19. Wojnicki I. A Rule-based Inference Engine Extending Knowledge Processing Capabilities of Relational Database Management Systems : дис. – Ph. D. Thesis), AGH University of Science and Technology, 2004.

20. DB-Engines Ranking [Электронный ресурс]. URL: https://db-engines.com/en/ranking (дата обращения: 21.03.2022).

21. PostgreSQL 14.2 Documentation [Электронный ресурс]. URL: https://www.postgresql.org/docs/14/index.html (дата обращения: 20.03.2022).

22. PostgreSQL Agility vs. Stability [Электронный ресурс]. URL: https://www.enterprisedb.com/blog/postgresql-agility-vs-stability (дата обращения: 23.06.2022).

23. Adventure Works Cycles Business Scenarios [Электронный ресурс]. URL: https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2008/ms124825(v=sql.100) (дата обращения: 28.03.2022).

24. Northwind Database [Электронный ресурс]. URL: https://github.com/pthom/northwind_psql (дата обращения: 28.03.2022).

25. Chinook Database [Электронный ресурс]. URL: https://github.com/lerocha/chinook-database (дата обращения: 28.03.2022).

26. SWI-Prolog documentation [Электронный ресурс]. URL: https://www.swi-prolog.org/pldoc/doc_for?object=root (дата обращения: 17.06.2022).

27. Docker overview | Docker Documentation [Электронный ресурс]. URL: https://docs.docker.com/get-started/overview/ (дата обращения: 08.05.2022).

28. Official Ubuntu Documentation [Электронный ресурс]. URL: https://help.ubuntu.com/lts/ubuntu-help/index.html (дата обращения 08.05.2022).

29. PostgreSQL Documentation: 14: Using EXPLAIN [Электронный ресурс]. URL: https://www.postgresql.org/docs/current/using-explain.html (дата обращения: 18.06.2022).

30. Abiteboul S., Vianu V. Datalog extensions for database queries and updates //Journal of Computer and System Sciences. – 1991. - Т. 43. – №. 1. – С. 62-124.

ПРИЛОЖЕНИЕ А.

Диаграммы БД.

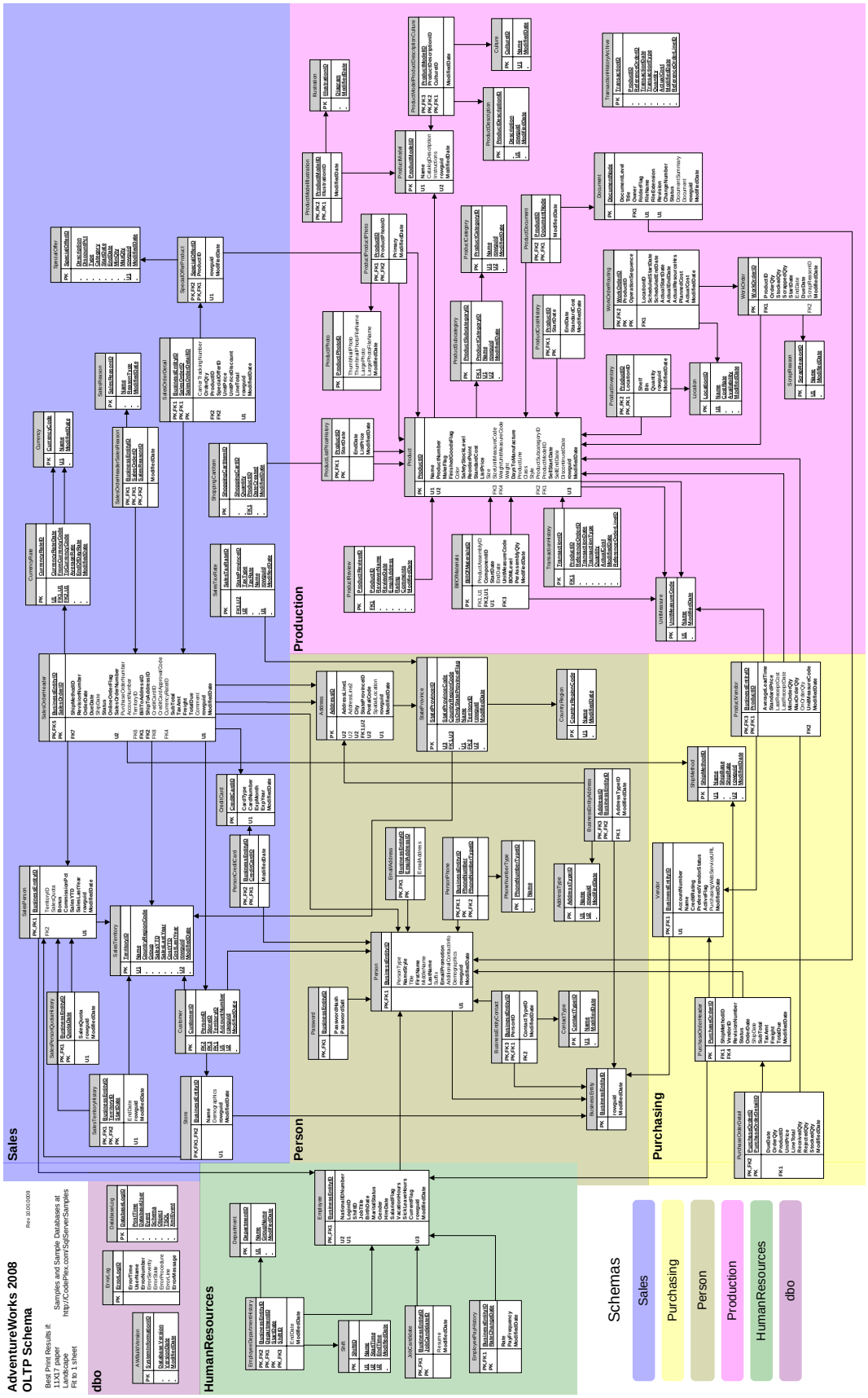

Рисунок А.1 – Диаграмма БД «AdventureWorks»

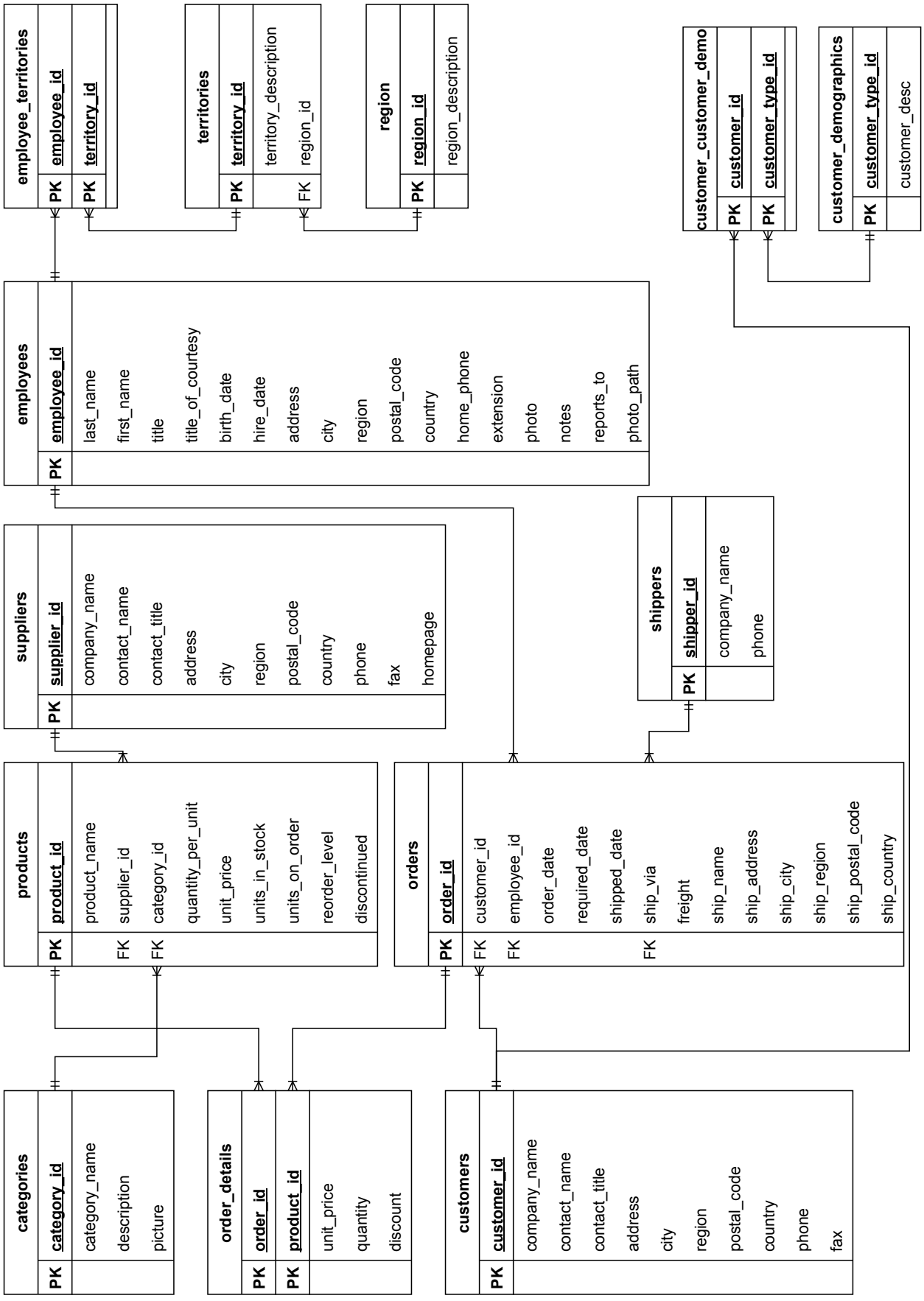

Рисунок А.2 – Диаграмма БД «Northwind»

Рисунок А.3 – Диаграмма БД «Chinook»

ПРИЛОЖЕНИЕ Б.
Развертывание приложения.

Листинг Б.1 – Сценарий запуска контейнеров с базами данных

```
docker-compose up -d;
# просмотр команд запуска
docker ps;

# подключение к контейнеру postgres
docker exec -it postgres bash;
su postgres;
psql;

# подключение к контейнеру swipl (команда запуска)
swipl;
```