Дипломная работа по профессии «SQL-разработчик»

Выполнила:

Мещерякова О.И.

Группа SQLD-9

Оглавление

Введение

Цель данной дипломной работы заключалась в разработке и внедрении серверного приложения для управления базой данных, что включало проектирование структуры БД, настройку конфигурационных параметров, а также проведение тестирования системы.

Основные задачи, которые решались в рамках работы:

1. Проектирование и создание структуры базы данных с использованием современных инструментов.
2. Настройка и оптимизация работы базы данных.
3. Отладка и обеспечение безопасности и доступности данных в системе.
4. Проведение тестирования системы для оценки её производительности и надёжности.

1. Описание структуры базы данных

1.1 Таблицы и поля.

Ниже представлена структура интернет-магазина:

1. City.

- id **int4**
- city **varchar**(50)

2. remaining_products.

- id **int4**
- id_product **int4**
- id_city **int4**
- quantity_in_stock **int4**

3. product.

- id **int4**
- product_name **varchar**(255
- product_category **varchar**(100)
- price **numeric**(10, 2)
- quantity **int4**

4. basket

- id **int4**
- id_buyer **int4**
- id_product **int4**
- product_quantity **int4**

5. customers

- id **int4**
- **"FIO" varchar**(100)
- address **varchar**(255)
- **"phone number" varchar**(20)
- email **varchar**(50)

6. reviews

- id **int4**
- id_product **int4**
- id_buyer **int4**
- rating **int4**

- **"comment" text**

7. orders

- id **int4**
- **"order date" date**
- **"order status" varchar**(50)
- id_buyer **int4**
- **CONSTRAINT "Заказы_pkey" PRIMARY KEY** (id)

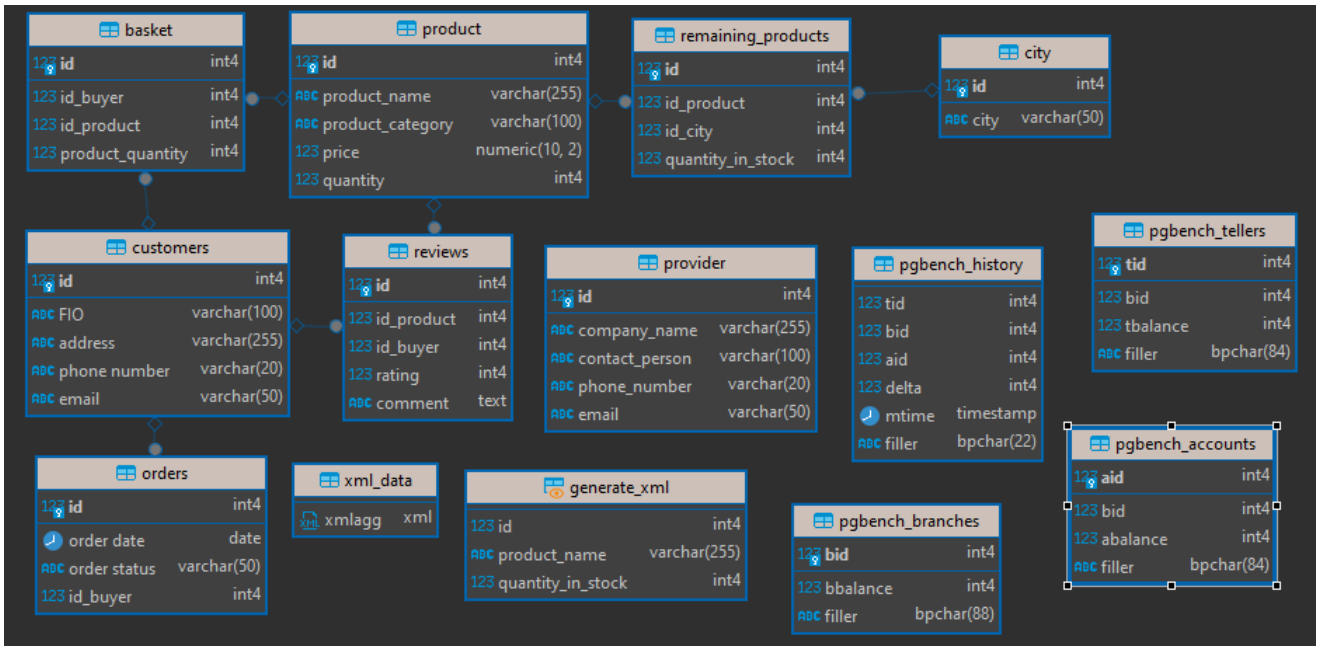
8. provider

- id **int4**
- company_name **varchar**(255)
- contact_person **varchar**(100)
- phone_number **varchar**(20)
- email **varchar**(50)

9. pgbench_branches

- bid **int4**
- bbalance **int4**
- filler **bpchar**(88)

1.2 ER-диаграмма:



2.Типы объектов в базе данных

2.1 Индексы.

По предполагаемому количеству заказов в сутки от 1000 до 5000 в разных городах, были созданы индексы исполняющие функции улучшения скорости выполнения запросов.

- **CREATE INDEX** idx_basket_id_customer **ON** public.basket **USING** btree (id_buyer);
- **CREATE INDEX** idx_product_category **ON** public.product **USING** btree (product_category);
- **CREATE INDEX** idx_customer_id **ON** public.customers **USING** btree (id);
- **CREATE INDEX** idx_reviews_id_product **ON** public.reviews **USING** btree (id_product);
- **CREATE INDEX** idx_order_data **ON** public.orders **USING** btree ("**order date**");
- **CREATE INDEX** idx_orders_id_customer **ON** public.orders **USING** btree (id_buyer);

2.2 Хранимые процедуры и функции.

Функции и хранимые процедуры в SQL, как и в любом другом языке программирования, обеспечивают возможность повторного использования и гибкость. В данной работе были созданы:

1. Добавление нового товара;
2. Обновление информации о товаре;
3. Удаление товара;
4. Оформление заказа;
5. Создание записи о заказе;
6. Добавление товаров в заказ;
7. Подсчёт общей стоимости заказа;
8. Процедура формирования актуального списка товаров с ненулевым остатком в XML-формате.

2.3 Роли.

Для создания ролей был написан скрипт на python, который генерирует случайный пароли и создаёт пользователя. После создания user присваивает к определённой роли. На данный момент grant для ролей были заполнены вручную. Ниже представлены роли (скрип)

1. marketer
2. analyst
3. designer
4. engineer
5. seospecialist
6. worker
7. thirdpartyservice
8. admin

2.4 Табличные представления.

Для выгрузки списка товаров в XML-формате было создано табличное представление generate_xml для вызова процедуры generate_and_download_xml

CREATE OR REPLACE VIEW generate_xml **AS**

SELECT r.id, p.product_name, r.quantity_in_stock

FROM public.remaining_products r

JOIN public.product p **ON** p.id = r.id_product;

3.Конфигурация.

3.1 Логирование.

SQL-операций:

- log_destination = ‘csvlog’ # Журнал будет записываться в формате CSV
- logging_collector = on # Включение сборщика записей
- log_directory = ‘pg_log’ # Каталог, в котором будет сохранен журнал (его необходимо предварительно создать)
- log_filename = ‘postgresql-%Y-%m-%d_%H%M%S.log’ # Шаблон имени файла журнала
- log_statement = ‘all’ # Журналирование всех SQL-операций
- log_min_duration_statement = 500 # Журналирует операции, выполняющиеся дольше указанного времени в миллисекундах.

Postgres:

- log_connections = on # Эта настройка определяет, будут ли журналироваться попытки установки соединения с базой данных.
- log_duration = on # Включение записи продолжительности выполнения каждого SQL запроса в журнал. Полезно для мониторинга и оптимизации производительности.
- log_line_prefix = '%m [%p] %q%u@%d ' #Данная настройка используется для форматирования каждой строки журнала.
- log_statement = 'ddl' #Эта настройка определяет какие операторы SQL будут журналироваться. Оператор "ddl" указывает журналировать только операции изменения схемы (Data Definition Language), такие как создание, изменение и удаление таблиц и индексов.
- log_timezone = 'Etc/UTC' #Устанавливает временную зону для журнальных записей. В данном случае, журнал будет отображать временную зону "Etc/UTC" (Скоординированное мировое время).

3.2 Тестирование.

Для настройки репликации и автоматического переключения при падении мастера с использованием репликационного менеджера repmgr использовался:

- Основной сервер (master): 192.168.68.105
- Реплицированный сервер (slave): 192.168.68.106

Шаг 1: Установка PostgreSQL и repmgr

Установила repmgr на обоих серверах:

- sudo apt-get update
- sudo apt install repmgr

Шаг 2: Основная настройка на мастере (192.168.68.105)

Настройка PostgreSQL

Отредактировала файл конфигурации postgresql.conf:

- Настройки указаны ниже в п. Архивирование.

Добавила следующие строки для разрешения подключения к slave в файл pg_hba.conf:

 # Allow replication connections from the slave server

- host replication admin 192.168.68.106/24 md5

Перезапустила PostgreSQL для применения изменений.

Настройка repmgr.conf

/etc/repmgr/15/repmgr.conf:

- node_id=1
- node_name='master'
- conninfo='host=192.168.68.105 user=admin dbname=Diplom connect_timeout=2'
- data_directory='/var/lib/postgresql/15/main'

Подключение к БД и пароли прописаны .pgss

- postgres@student:~$ repmgr -f /etc/repmgr/15/repmgr.conf primary register
- INFO: connecting to primary database...
- NOTICE: attempting to install extension "repmgr"

- NOTICE: "repmgr" extension successfully installed
- NOTICE: primary node record (ID: 1) registered

Шаг 3: Настройка слейва (192.168.68.106)

1. **Настройка реплики.**

    - vi postgresql.conf:
    - listen_addresses = '*'
    - hot_standby = on

Добавила следующие строки для разрешения подключения к master в файл pg_hba.conf:

 # Allow replication connections from master and repmgr connections

    - host replication replicator 192.168.68.105/32 md5
    - host all all 0.0.0.0/0 md5

2. **Клонирование мастера.**

    - repmgr -h 192.168.68.105 -U admin -d Diplom -f /etc/repmgr/15/repmgr.conf standby clone --force
    - sudo systemctl start postgresql

3. **Регистрация реплики**

    - sudo -i -u postgres
    - repmgr -f /etc/repmgr/15/repmgr.conf standby register

Шаг 4: Настройка автоматического переключения

Для автоматического failover на обоих серверах настраивается repmgrd:

    - failover=automatic
    - promote_command='repmgr standby promote -f /etc/repmgr/15/repmgr.conf --log-to-file'
    - follow_command='repmgr standby follow -f /etc/repmgr/15/repmgr.conf --log-to-file --upstream-node-id=%n'

Запуск repmgrd на каждом сервере:

    - sudo -i -u postgres
    - repmgrd -f /etc/repmgr/15/repmgr.conf –daemonize

Шаг 5: Проверка настройки

Проверка статуса репликации и установки кластеров:

- repmgr cluster show

Role | Name | Upstream | Connection String

---- | ---- | -------- | -----------------

* master | node1 | |

standby | node2 | node1 | host=192.168.68.105 user=admin dbname=Diplom connect_timeout=2

3.3 Архивирование.

Прогнозирование требуемого места на диске для хранения резервных копий и WAL-логов может зависеть от нескольких факторов, таких как объем данных, кратность резервирования и политика хранения. Вот примерный прогноз, основанный на предоставленной информации:

1. Ежедневные данные заказов:

Предположим, что среднее количество заказов составляет 3000 в день. Если каждый заказ имеет средний размер в 1 МБ (включая все прикрепленные файлы, изображения, детали заказа и т.д.), то ежедневно будет создаваться 3000 МБ (или 3 ГБ) новых данных.

2. Резервные копии:

Для интернет-магазина важно регулярно создавать резервные копии для обеспечения безопасности данных. Рекомендуется делать регулярные полные бэкапы базы данных, а также инкрементальные бэкапы WAL-логов.

Частота создания резервных копий может быть задана каждые 24 часа. Размер полной резервной копии будет примерно равен размеру данных, с учетом приращений внутри этого периода времени. Размер инкрементальных бэкапов будет зависеть от объема изменений в базе данных, которые произошли с момента предыдущего бэкапа.

Предположим, что размер полной резервной копии равен 3 ГБ (см. пункт 1) и размер инкрементального бэкапа WAL-логов составляет 100 МБ в сутки. При создании ежедневного полного бэкапа и ежедневного инкрементального бэкапа в течение года (365 дней), общий размер резервных копий будет примерно равен:

= (3 ГБ * 365) + (100 МБ * 365)
= 1095 ГБ + 36.5 ГБ
≈ 1131.5 ГБ или около 1.1 ТБ

3. Хранение и удаление устаревших копий и WAL-логов:

Важно установить политику хранения и удаления старых резервных копий и WAL-логов в соответствии с требованиями вашего бизнеса и сохранности данных.

Рекомендуется сохранять несколько последних полных копий, чтобы можно было восстановить данные в случае повреждения или потери. Удаление устаревших копий может осуществляться, например, через 1 месяц (настроены 40 дней) после их создания.

WAL-логи также можно хранить определенное время. Они используются для восстановления данных до определенного момента времени. После завершения символьного восстановления к точке во времени, соответствующей последней резервной копии, устаревшие WAL-логи могут быть удалены.

Важно сконфигурировать Politicy-Based Retention (PBR) для управления удалением старых копий и WAL-логов в соответствии с вашими требованиями по сохранности данных и пространству на диске.

Настройка конфигурации WAL archive:

wal_level = archive

fsync = on

wal_buffers = 32MB

max_wal_size = 1GB

min_wal_size = 80MB

archive_mode = on

archive_command = archive_command = 'rsync -av %p /home/archive/%f && find /home/archive -type f -name "*.gz" -mtime +40 -delete'

Автоматическое удаление python:

```python
import os
import datetime

archive_dir = '/home/archive/'
delete_date = datetime.datetime.now() - datetime.timedelta(days=40)
files = os.listdir(archive_dir)
for file_name in files:
        file_path = os.path.join(archive_dir, file_name)
        creation_time = datetime.datetime.fromtimestamp(os.path.getctime(file_path))
        if creation_time < delete_date:
            os.remove(file_path)
```

Данный скрипт можно добавить в планировщик задач такой как cron, чтобы он выполнял регулярно. Так же требуется настроить права доступа к директории архива, чтобы пользователь имел доступ для удаления файлов.

3.4 Резервное копирование.

Для резервного копирования используется pg_dump. Эта команда создаст резервную копию базы данных Diplom и сохранит её в файл backup.sql

- pg_dump -h 192.168.68.105 -U admin -d Diplom -w > ./backup/backup_$(date +"%Y-%m-%d").sql

Данное копирование происходит автоматически, с помощью планировщика задач cron.

Заключение.

В соответствии с представленными бизнес-требованиями, задача состояла в разработке и реализации базы данных для интернет-магазина, применяя современные практики при её развёртывании и учитывая специфические потребности бизнеса.

Первоначально, необходимо обеспечить масштабируемость базы данных и поддержку высокой производительности. Учитывая ожидаемое количество заказов в диапазоне от 1 000 до 5 000 в сутки в разных городах и количество зарегистрированных клиентов превышающее 100 000 человек, была разработана архитектура, способная эффективно обрабатывать такую большую нагрузку. Важно оптимизировать SQL-запросы и использовать подходящие индексы и структуры таблиц с целью минимизации времени выполнения запросов и обеспечения высокоскоростной обработки множества запросов.

Безопасность и отказоустойчивость также являются важными аспектами. Реализована отказоустойчивая архитектура базы данных, которая исключает возможность простоев по различным причинам, таким как падение серверов, обслуживание и другие. Регулярное создание бэкапов данных также необходимо для обеспечения надежности и возможности восстановления системы. Не менее важным является логирование всех SQL-операций, выполняющихся более 500 мс. Такое логирование позволит отслеживать и анализировать производительность запросов и операций в базе данных, выявлять возможные проблемы и оптимизировать работу системы.

Важно было предусмотреть разграничение прав доступа для различных категорий пользователей, таких как сторонние сервисы, маркетологи, аналитики и SEO-специалисты. Разработка ролей и привилегий для каждой категории поможет обеспечить безопасный доступ с использованием SSL.

Для обеспечения эффективной работы магазина, был создан механизм формирования актуального списка товаров с ненулевым остатком в XML-формате. Это позволит предоставлять актуальную информацию о наличии товаров.

В результате проведенной работы удалось достичь:

1. Создания эффективной структуры базы данных, включая таблицы, индексы, последовательности, представления, функции и хранимые процедуры.
2. Настройки логирования и мониторинга базы данных.
3. Организации процесса резервного копирования и восстановления данных.
4. Разработки механизма переключения реплики базы данных при отказе основного сервера.

В заключении, разработка базы данных для интернет-магазина требует соблюдения современных стандартов и практик, учета специфики бизнеса и потребностей пользователей. Результатом должна стать гибкая, масштабируемая и безопасная база данных, способная обеспечивать высокую производительность и эффективную работу интернет-магазина в соответствии с поставленными требованиями.