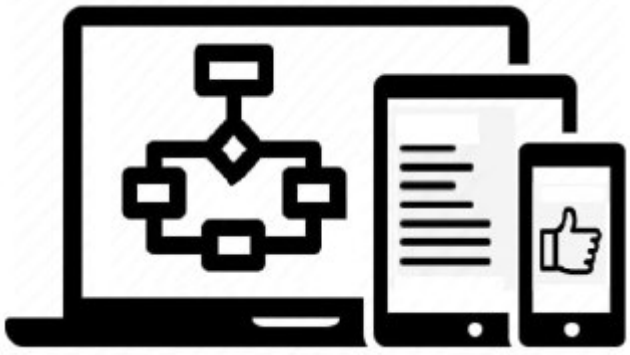
МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Программное обеспечение информационных технологий»

СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

Методические рекомендации
к практическим занятиям
для студентов специальности
1-53 01 02 «Автоматизированные системы
обработки информации»
дневной и заочной форм обучения



Могилев 2023

УДК 004.65
ББК 32.973.26-0.18.2
С89

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Программное обеспечение информационных технологий» «26» апреля 2023 г., протокол № 10

Составители: канд. техн. наук, доц. К. В. Захарченков;
канд. техн. наук, доц. Т. В. Мрочек

Рецензент Ю. С. Романович

Методические рекомендации содержат описание хода выполнения практических работ, выполняемых во втором семестре изучения дисциплины «Системы управления базами данных». Рассматриваются основы работы с Microsoft SQL Server и языком Transact-SQL.

Учебное издание

СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

Ответственный за выпуск	В. В. Кутузов
Корректор	Т. А. Рыжикова
Компьютерная верстка	Е. В. Ковалевская

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 21 экз. Заказ №

Издатель и полиграфическое исполнение:
Межгосударственное образовательное учреждение высшего образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 07.03.2019.
Пр-т Мира, 43, 212022, г. Могилев.

Содержание

Введение

Целью учебной дисциплины «Системы управления базами данных» (СУБД) является формирование профессиональных компетенций для работы с современными системами управления базами данных, основами эксплуатации баз данных (БД) в составе автоматизированных систем обработки информации, внедряемых в различных областях науки, техники и экономики.

В результате освоения учебной дисциплины студент:

а) ознакомится с:

– языком SQL;

– способами работы с реляционными базами данных;

– способами реализации распределенной и параллельной обработки данных;

– возможностями современных СУБД для построения систем поддержки принятия решений;

б) научится:

– практически создавать и администрировать базы данных;

– создавать интерфейс с базами данных;

– организовывать работу в многопользовательской базе данных;

– устанавливать и конфигурировать серверные и клиентские приложения баз данных;

в) овладеет:

– методами, средствами и технологиями разработки информационных моделей и их программной реализации в выбранной СУБД;

– методами программирования систем на основе баз данных.

Методические рекомендации содержат описание четырех практических работ», выполняемых при изучении дисциплины «Системы управления базами данных», задания, контрольные вопросы, список литературы [1–11], которую необходимо использовать при подготовке к защите практических работ.

1 Язык SQL. Работа с триггерами

Цель: научиться создавать триггеры на T-SQL.

Теоретические положения

Триггер – это специальный тип хранимых процедур, который запускается автоматически на стороне сервера при выполнении тех или иных действий с данными таблицы. Каждый триггер привязывается к конкретной таблице [4, 11].

Напрямую обратиться к триггеру нельзя. Он вызывается автоматически при наступлении соответствующего события в БД – добавление новой строки в таблицу, изменение или удаление строки таблицы. Триггер может срабатывать, когда соответствующее действие с БД выполняет клиентское приложение, хранимая процедура или триггер (другой или тот же самый).

Триггеры DML вызываются при выполнении операторов INSERT, UPDATE или DELETE. Можно указать время вызова триггера:

– AFTER – триггер срабатывает только после успешного запуска инструкции DML, запустившей триггер. Синонимом в синтаксисе оператора создания триггера является FOR. Также до запуска триггера должны успешно завершиться все каскадные действия и проверки ограничений, на которые есть ссылки. Триггеры AFTER нельзя определить для представлений;

– INSTEAD OF – триггер вызывается вместо действий, заданных оператором INSERT, UPDATE или DELETE, т. е. переопределяет действия запускающих инструкций. Аргумент INSTEAD OF нельзя использовать для триггеров DDL. Для каждой инструкции INSERT, UPDATE или DELETE в таблице или представлении можно определить не более одного триггера INSTEAD OF.

При выполнении триггеров DML создаются две виртуальных таблицы, которые содержат данные, на которые влияет выполнение триггера. Эти таблицы называются inserted и deleted, и они имеют ту же структуру, что и структура таблицы, для которой определен триггер. При добавлении данных (INSERT) в триггере можно получить добавленные данные из виртуальной таблицы inserted. При удалении все удаленные данные помещаются в виртуальную таблицу deleted. При выполнении операции UPDATE используются deleted и inserted.

Триггеры DDL активируются в ответ на разные события языка описания данных DDL. Эти события прежде всего соответствуют инструкциям Transact-SQL CREATE, ALTER, DROP и некоторым системным хранимым процедурам, которые выполняют схожие с DDL операции.

Для создания триггера DML используется следующая инструкция T-SQL.

```
CREATE TRIGGER  [ schema_name . ] Trigger_name
ON Table_name | View_name
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
AS
sql_statement [...n] }
```

Trigger_name – задает имя триггера, с помощью которого он будет распознаваться хранимыми процедурами и командами T-SQL. Имя триггера должно быть уникальным в пределах БД. Оно должно характеризовать действие, выполняемое им, записываться в формате <глагол_объект> и содержать в себе имя таблицы и название момента срабатывания.

Table_name | View_name – имя таблицы БД (или представления), к которой будет привязан триггер.

[DELETE] [,] [INSERT] [,] [UPDATE] – определяет инструкции изменения данных, при применении которых к таблице или представлению срабатывает триггер DML. При создании триггера должно быть указано хотя бы одно из этих ключевых слов, и допускается создание триггера, реагирующего на две или три команды в любом сочетании.

sql_statement – определяет набор команд, которые будут выполняться при запуске триггера.

Одно и то же действие триггера может быть определено более чем для одного действия пользователя (например, INSERT и UPDATE) в одной и той же инструкции CREATE TRIGGER.

Триггеры INSTEAD OF DELETE и INSTEAD OF UPDATE нельзя определить для таблицы, у которой есть внешний ключ с каскадным действием для операции DELETE или UPDATE.

Если в триггере происходит присвоение переменной, следует использовать инструкцию SET NOCOUNT в начале триггера, чтобы предотвратить возвращение каких-либо результирующих наборов.

Далее приведен пример триггера, который будет запрещать удаление записей таблицы «Issuing_books», если текущий пользователь не владелец базы данных и если поле «дата выдачи» содержит какое-либо значение.

```
CREATE TRIGGER Ondelete_issuing_books   /* Объявляем имя триггера */
ON Issuing_books                /* имя таблицы, с которой связан триггер */
FOR DELETE /*Указываем операцию, на которую будет срабатывать триггер (здесь на  удаление)*/
AS
IF ( SELECT COUNT(*)        /*проверяет*/
FROM Issuing_books          /*записи из таблицы «Issuing_books»*/
WHERE Issuing_books.ib_date_of_issue IS NOT NULL) > 0   /*условие проверяет наличие записи в поле «ib_date_of_issue». Если COUNT(*) возвращает значение, отличное от нуля (т. е. запись есть), то первое условие IF не выполнено*/
AND (CURRENT_USER <> 'dbo')        /*вызывается функция определения имени текущего пользователя и проверяется, владелец ли он*/
BEGIN
PRINT 'у вас нет прав на удаление этой записи'        /*выдача сообщения о неудаче операции*/
ROLLBACK TRANSACTION      /*откат (отмена) транзакции*/
END
```

Задание

В разрабатываемой БД необходимо реализовать 10 триггеров, реагирующих на различные команды (INSERT, DELETE, UPDATE) и содержащих не менее 10 различных стандартных функций SQL.

Содержание отчета: тема и цель работы; SQL-код 10 триггеров.

Контрольные вопросы

1 Что такое триггер? Какие типы триггеров различают?
2 Как создать триггер?
3 С помощью каких команд T-SQL можно изменить или удалить триггер?
4 Чем отличается триггер от хранимой процедуры?
5 Для чего нужны виртуальные таблицы inserted и deleted?

2 Создание индексов средствами SQL

Цель: получить навыки определения индексируемых столбцов таблиц.

Теоретические положения

Индекс представляет собой дополнение к таблице, помогающее ускорить поиск необходимых данных за счет физического или логического их упорядочивания. Он является набором ссылок, упорядоченным по определенному (индексируемому) столбцу таблицы. Физически индекс представляет собой упорядоченный набор значений из индексированного столбца с указателями на места физического размещения исходных строк в структуре базы данных. В индексе хранится не информация обо всей строке данных, а лишь ссылка на нее. Использование индексов позволяет избежать полного сканирования таблицы. На сегодняшний день существует большое количество разновидностей индексов, основными из которых являются кластерный, некластерный, уникальный, покрывающий, полнотекстовый. Более подробно они описаны в [4, 6, 10, 11].

При выборе столбца для индекса следует проанализировать, какие типы запросов чаще всего выполняются и какие столбцы являются ключевыми.

Ключевые столбцы – это такие колонки, которые задают критерии выборки данных, например порядок сортировки. Не следует индексировать столбцы, которые только считываются и не играют никакой роли в определении порядка выполнения запроса. Не следует индексировать слишком длинные колонки, например колонки с адресами или названиями компаний, достигающие длины несколько десятков символов. При необходимости можно создать укороченный вариант такого столбца, выбрав из него до десяти первых символов, и индексировать его. Индексирование длинных столбцов может существенно снизить производительность работы сервера. Индекс является самостоятельным объектом БД, но он связан с определенным столбцом таблицы.

Индексы создаются в следующих случаях:

– если операции чтения из таблицы выполняются гораздо чаще, чем операции модификации;

– если поле (или совокупность полей) часто используется в запросах в разделе WHERE;

– если нужно обеспечить уникальность значения поля (или совокупности полей), не являющегося первичным ключом (в этом случае создается уникальный индекс);

– если поле (или совокупность полей) является внешним ключом (так как в таком случае индексы могут существенно ускорить выполнение запросов с соединениями JOIN).

Формат команды CREATE INDEX на T-SQL имеет вид:

```
CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED]
INDEX index_name
ON table_or_view_name ( column [...n] )
[ INCLUDE ( column_name [ ,...n ] ) ]
[ WITH { PAD_INDEX = { ON | OFF } | FILLFACTOR = f
    | IGNORE_DUP_KEY } [ ,...n ] ) ]
```

При указании ключевого слова UNIQUE будет создан уникальный индекс, гарантирующий уникальность значений в индексируемом столбце. При создании такого индекса сервер выполняет предварительную проверку столбца на уникальность значений. Если в столбце есть хотя бы два одинаковых значения, индекс не создается. В индексируемом столбце также желательно запретить хранение значений NULL. После того как для столбца создан уникальный индекс, сервер не разрешает выполнение команд INSERT и UPDATE, которые приведут к появлению дублирующихся значений. Уникальный индекс является своеобразной надстройкой и может быть реализован как для кластерного, так и для некластерного индексов. В одной таблице могут существовать один уникальный кластерный индекс и множество уникальных некластерных индексов.

CLUSTERED – создаваемый индекс будет кластерным, т. е. при его определении в таблице физическое расположение данных перестраивается в соответствии со структурой индекса. Информация об индексе и сами данные физически располагаются вместе. Кластерным может быть только один индекс в таблице. Кластерный индекс создается до создания любых некластерных, т. к. при создании кластерного индекса все существующие некластерные индексы таблицы перестраиваются. Если аргумент CLUSTERED не указан, создается некластерный индекс. В качестве кластерного индекса следует выбирать наиболее часто используемые столбцы. Следует избегать создания кластерного индекса для часто изменяемых столбцов, т. к. сервер должен будет выполнять физическое перемещение всех данных в таблице, чтобы они находились в упорядоченном состоянии, как того требует кластерный индекс.

NONCLUSTERED – создаваемый индекс будет некластерным, и, в отличие от кластерных, он не перестраивает физическую структуру таблицы, а лишь организует ссылки на соответствующие строки. Каждая таблица может содержать до 999 некластерных индексов независимо от способа их создания: неявно с помощью ограничений PRIMARY KEY и UNIQUE или явно с помощью инструкции CREATE INDEX. Однако лучше ограничиться четырьмя-пятью индексами.

index_name – имя индекса, по которому он будет распознаваться командами Transact-SQL. Имя индекса должно быть уникальным в пределах таблицы.

table_or_view_name(column [...n]) – имя таблицы или представления, в которой содержатся один или несколько индексируемых столбцов. В скобках указываются имена столбцов, на основе которых будет построен индекс. Не допускается построение индекса на основе столбцов с типом данных ntext, text, varchar(max), nvarchar(max), varbinary(max), xml или image. Если указывается несколько столбцов, то создаваемый индекс будет составным. В один составной индекс можно включить до 16 столбцов.

Параметр INCLUDE позволяет создать покрывающий индекс, т. е. некластерный индекс, содержащий в своих листовых узлах информацию из дополнительного неключевого поля, не использующегося при создании самого индекса.

В предложении WITH перечисляются параметры создаваемого индекса.

Коэффициент заполнения FILLFACTOR = f задает заполнение в процентах каждой страницы индекса во время его первоначального создания или пересоздания. Значение f можно установить в диапазоне от 1 до 100 (по умолчанию f = 0). Значения коэффициентов заполнения 0 и 100 идентичны. Если f равен 100, компонент Database Engine создает индексы с полностью заполненными страницами конечного уровня. Чем больше значение f, тем меньше свободного места в страницах листьев индекса. Например, при значении f = 60 каждая страница листьев индекса будет иметь 40 % свободного места для вставки строк индекса в дальнейшем. Производительность операций считывания снижается обратно пропорционально значению коэффициента заполнения. Для редко изменяемых таблиц рекомендуется принимать f = 100; для часто изменяемых таблиц f = 50…70; в случае промежуточной ситуации f = 80…90 [4].

Параметр PAD_INDEX = {ON | OFF} задает возможность использования разреженного индекса. ON – допустим разреженный индекс, OFF (значение по умолчанию) – недопустим. В случае разреженного индекса должен присутствовать и коэффициент заполнения FILLFACTOR, задающий процент разрежения.

Параметр IGNORE_DUP_KEY = { ON | OFF } определяет ответ на ошибку, случающуюся, когда операция вставки пытается вставить в уникальный индекс повторяющиеся значения ключа. Применяется только к операциям вставки, производимым после создания или перестроения индекса. Параметр не работает во время выполнения инструкции CREATE INDEX, ALTER INDEX или UPDATE. Значение по умолчанию – OFF, которое означает, что, если в уникальный индекс вставляются повторяющиеся значения ключа, выводится сообщение об ошибке и будет выполнен откат всей операции INSERT. Значение ON означает, что, если в уникальный индекс вставляются повторяющиеся значения ключа, выводится

предупреждающее сообщение и с ошибкой завершаются только строки, нарушающие ограничение уникальности.

Для удаления индекса используется команда DROP INDEX, имеющая следующий синтаксис: DROP INDEX 'table.index' [...n]. Аргумент 'table.index' определяет удаляемый индекс в таблице.

Задание

Необходимо обосновать выбор столбцов таблиц для создания индексов в разрабатываемой БД и создать пять индексов для таблиц БД.

Содержание отчета: тема и цель работы; прокомментированный SQL-код выполнения задания с обоснованием выбора столбцов для индексации.

Контрольные вопросы

1 Что такое кластерный, некластерный, уникальный, составной и покрывающий индексы?

2 Какие критерии учитываются при выборе столбцов для индексирования?

3 С помощью каких команд T-SQL задаются различные виды индексов?

4 Перечислить общие рекомендации при планировании стратегии индексирования (назвать не менее восьми рекомендаций).

5 Когда использование индексов нецелесообразно?

3 Назначение прав доступа пользователям к объектам базы данных средствами SQL

Цель: изучить основы управления правами доступа к объектам базы данных в СУБД MS SQL Server.

Теоретические положения

Вопросы управления правами доступа к объектам БД подробно описаны в [1, 5, 9–11]. Здесь же рассмотрен SQL-код основных инструкций.

С каждым защищаемым объектом в SQL Server связаны разрешения, которые могут быть предоставлены участнику. Управление разрешениями в компоненте Database Engine осуществляется на уровне сервера (назначение разрешений именам входа и ролям сервера) и на уровне базы данных (назначение разрешений пользователям и ролям базы данных).

Для предоставления разрешений permission на защищаемый объект securable участнику principal используется инструкция GRANT, общая концепция которой имеет следующий вид: GRANT <some permission> ON <some object> TO <some user, login, or group>.

Синтаксис инструкции GRANT:

```
GRANT { ALL [ PRIVILEGES ] }   | permission [ ( column [ ,...n ] ) ] [ ,...n ]
    [ ON [ class :: ] securable ] TO principal [ ,...n ]
    [ WITH GRANT OPTION ] [ AS principal ]
```

Инструкция DENY запрещает разрешение для участника. Предотвращает наследование разрешения участником через его членство в группе или роли. DENY имеет приоритет над всеми разрешениями, но не применяется к владельцам объектов или членам с предопределенной ролью сервера sysadmin (членам предопределенной роли сервера sysadmin и владельцам объектов не может быть отказано в разрешениях). Синтаксис инструкции DENY:

```
DENY   { ALL [ PRIVILEGES ] } | <permission>  [ ( column [ ,...n ] ) ] [ ,...n ]
   [ ON [ <class> :: ] securable ]
   TO principal [ ,...n ]
   [ CASCADE] [ AS principal ]  [;]
```

Инструкция REVOKE удаляет разрешение, выданное или запрещенное ранее (иногда говорят, что REVOKE используется для неявного отклонения доступа к объектам БД):

```
REVOKE [ GRANT OPTION FOR ]
    {   [ ALL [ PRIVILEGES ] ]   |   permission [ ( column [ ,...n ] ) ] [ ,...n ]  }
    [ ON [ class :: ] securable ]
    { TO | FROM } principal [ ,...n ]
    [ CASCADE] [ AS principal ]
```

Охарактеризуем назначение параметров инструкций.

ALL – этот параметр устарел и сохранен только для поддержки обратной совместимости. Он не предоставляет все возможные разрешения (см. документацию https://docs.microsoft.com/). Вместо ALL следует предоставлять конкретные разрешения.

PRIVILEGES – включено для обеспечения совместимости с требованиями ISO. Не изменяет работу ALL.

permission – имя разрешения, которое предоставляется пользователю. Можно предоставлять одновременно несколько разрешений.

column – указывает имя столбца таблицы, на который предоставляется разрешение.

class – указывает класс защищаемого объекта, для которого предоставляется разрешение. Квалификатор области :: является обязательным.

securable – указывает защищаемый объект, на который предоставляется разрешение.

TO principal – имя участника. Состав участников, которым можно предоставлять разрешения, меняется в зависимости от защищаемого объекта.

WITH GRANT OPTION – позволяет пользователю, получающему разрешение, получить также возможность предоставлять данное разрешение другим участникам.

AS principal – указывает, что участник, записанный как предоставляющий разрешение, должен быть участником, отличным от пользователя, выполняющего инструкцию. Предположим, что пользователь Ольга – это участник 12, пользователь Павел – участник 15. Ольга выполняет GRANT SELECT ON OBJECT::X TO Steven WITH GRANT OPTION AS Paul;. В таблице sys.database_permissions для параметра grantor_principal_id будет указано значение 15 (Павел), хотя инструкция была выполнена пользователем 12 (Ольга). Использовать предложение AS обычно не рекомендуется, за исключением необходимости явного определения цепочки разрешения.

GRANT OPTION FOR – указывает, что возможность предоставлять указанное разрешение будет отменена. Данный аргумент необходим при использовании аргумента CASCADE. Если участник обладает указанным разрешением без параметра GRANT, будет отменено само разрешение.

TO | FROM principal – имя участника. Участники, у которых может быть отменено разрешение на доступ к защищаемому объекту, различны.

CASCADE – позволяет отзывать права не только у данного пользователя, но и у всех пользователей, которым он предоставил данные права. Аргумент CASCADE необходимо использовать совместно с GRANT OPTION FOR. Каскадная отмена разрешения, предоставленного с помощью WITH GRANT OPTION, приведет к отмене разрешений GRANT и DENY для этого разрешения.

Предоставление разрешения удаляет DENY или REVOKE для этого разрешения на данный защищаемый объект. Если то же разрешение запрещено для более высокой области действия, которая содержит данный защищаемый объект, то DENY имеет более высокий приоритет. Однако отмена разрешения на более высоком уровне не имеет более высокого приоритета.

Задание

Для разрабатываемой БД необходимо реализовать систему безопасности: создать одну роль и трех пользователей. В созданную роль нужно включить двух пользователей. Третьего пользователя следует включить в одну из стандартных ролей SQL Server. Роли или пользователи должны иметь наименования, указанные в техническом задании на разработку базы данных информационной системы, реализуемой в курсовом проектировании.

Далее необходимо назначить права на все ранее разработанные объекты БД. Следует обратить внимание, что не должно быть объектов базы данных, на которые ни у кого из пользователей прав нет. В отчете указывается, с какими объектами базы данных позволяет пользователям работать каждая роль, а также приводятся фрагменты инструкций SQL, позволяющих создать роли, пользователей и предоставить этим пользователям права доступа к объектам базы данных.

Назначение прав доступа на объекты базы данных (таблицы, представления, хранимые процедуры, курсоры) должно быть проиллюстрировано таблицей 3.1 с помощью общепринятых сокращений, произошедших от аббревиатуры

CRUD – Create, Read, Update, Delete (C – право на создание записи, R – право на чтение записи, U – право на обновление записи, D – право на удаление записи).

Таблица 3.1 – Перечень прав пользователей (категорий пользователей) относительно объектов базы данных

Имя объекта базы данных	Пользователь 1	Пользователь 2	Пользователь *N*	Приложение	Гость	Администратор
Table1	CRU	R	U	R	–	CRUD
View1	R	R	CRU	R	R	CRUD

Содержание отчета: тема и цель работы; описание распределения прав доступа пользователей; SQL-код выполнения задания.

Контрольные вопросы

1 На какие категории можно разделить права в SQL Server?

2 Перечислить стандартные роли сервера и базы данных.

3 Какие команды T-SQL используются для предоставления прав доступа, запрещения и неявного отклонения доступа?

4 Оконные функции

Цель: изучить оконные, аналитические, ранжирующие функции и функции смещения в T-SQL.

Теоретические положения

Оконная функция (window function) – инструмент для выполнения вычислений над группой строк (окном, секцией, партицией) в таблице, объединенных общим признаком, например, идентификатором клиента [3].

Окно из набора строк – это группа строк, указанная для оконной функции по одному или нескольким столбцам таблицы. Каждая оконная функция в запросе может иметь свои собственные секции, которые могут быть разделены по различным столбцам таблицы. Это позволяет производить оконные вычисления только над определенными группами строк, что обеспечивает более точные результаты и более гибкий анализ данных.

Оконные функции похожи на агрегатные функции, но в отличие от них оконные функции не объединяют несколько строк в одну. Вместо этого каждая строка продолжает существовать отдельно, но результат работы оконных функций добавляется к выходным данным как дополнительное поле. Оконные функции не приводят к группированию строк в одну строку вывода, строки сохраняют

свои отдельные идентификаторы, а агрегированное значение добавляется к каждой строке.

Окно определяется с помощью обязательной инструкции OVER(), имеющей следующий синтаксис:

```
SELECT
Название функции (столбец для вычислений)
OVER (
    PARTITION BY столбец для секционирования
    ORDER BY столбец для сортировки
    ROWS | RANGE выражение для ограничения строк в пределах группы
    )
```

Охарактеризуем назначение параметров данной инструкции.

PARTITION BY разделяет результирующий набор запроса на секции. Оконная функция применяется к каждой секции отдельно, и вычисление начинается заново для каждой секции.

ORDER BY определяет сортировку строк в каждой секции результирующего набора.

ROWS или RANGE ограничивает строки в пределах секции, указывая начальную и конечную точки, и требует аргумента ORDER BY. По умолчанию будет охватывать интервал от начала секции до текущего элемента, если указан аргумент ORDER BY.

В примерах, рассматриваемых далее, будет использоваться таблица 4.1.

Таблица 4.1 – Данные таблицы Product

ID	category	product	price	date
1	Мышь	Xiaomi Mi Wireless	19.99	2023-05-10
2	Мышь	Logitech G Pro	39.99	2023-05-10
3	Мышь	FANTECH X9 Thor	25.55	2023-05-12
4	Клавиатура	Redragon K552	34.99	2023-05-12
5	Клавиатура	Delux T9	29.99	2023-05-12

Рассмотрим пример использования инструкции PARTITION BY, которая определяет столбец, по которому будет производиться группировка, и является ключевой в разделении набора строк на окна. Инструкция сгруппирует строки по полю order_date и для каждой группы рассчитает среднюю цену, записанную в дополнительном столбце в таблице 4.2.

```
SELECT
   id, category, product, price, order_date,
   AVG(price)  OVER(PARTITION BY order_date) AS 'AVG_price'
FROM Products
```

Таблица 4.2 – Результаты использования инструкции PARTITION BY

ID	category	product	price	order_date	avg_price
1	Мышь	Logitech G Pro	39.99	2023-05-10	29.990000
2	Мышь	Xiaomi Mi Wireless	19.99	2023-05-10	29.990000
3	Мышь	FANTECH X9 Thor	25.55	2023-05-12	30.176666
4	Клавиатура	Redragon K552	34.99	2023-05-12	30.176666
5	Клавиатура	Delux T9	29.99	2023-05-12	30.176666

Оконные функции разделяются на **агрегатные**, **ранжирующие**, **аналитические** или **статистические** (функции распределения) и **функции смещения**.

Агрегатные функции MAX(), MIN(), SUM, AVG(), COUNT() можно применить к секциям результирующего набора запроса. Синтаксис функций имеет следующий вид:

```
MAX | MIN | SUM | AVG | COUNT(<столбец>)
OVER ( [ PARTITION BY <столбцы> ]
[ ORDER BY <столбцы> ])
```

Отсортируем значения внутри каждой секции с помощью ORDER BY и для каждой секции найдем среднее и минимальное значения (результаты выполнения этого запроса показаны в таблице 4.3):

```
SELECT
   ID, Category, Product, Price, order_date,
   AVG(Price) OVER(PARTITION BY order_date) AS 'avg_price',
   MIN(Price) OVER(PARTITION BY order_date ORDER BY Price) AS
'min_price'
FROM Products
```

Таблица 4.3 – Результаты использования PARTITION BY, ORDER BY, AVG() и MIN()

ID	category	product	price	order_date	avg_price	min_price
2	Мышь	Xiaomi Mi Wireless	19.99	2023-05-10	29.990000	19.99
1	Мышь	Logitech G Pro	39.99	2023-05-10	29.990000	19.99
3	Мышь	FANTECH X9 Thor	25.55	2023-05-12	30.176666	25.55
5	Клавиатура	Delux T9	29.99	2023-05-12	30.176666	25.55
4	Клавиатура	Redragon K552	34.99	2023-05-12	30.176666	25.55

В агрегатных функциях предложение PARTITION BY можно не указывать, тогда функция обрабатывает все строки результирующего набора запроса как одну группу. Необязательное предложение ORDER BY определяет последовательность, в которой строкам назначаются уникальные номера.

Ранжирующие функции ранжируют значение для каждой строки в секции (например, нумеруют строки по группам или выставляют ранг для рейтинга) и являются недетерминированными.

В ранжирующих функциях предложение PARTITION BY можно не указывать, тогда функция обрабатывает все строки результирующего набора запроса как одну группу. Обязательное предложение ORDER BY определяет последовательность, в которой строкам назначаются уникальные номера.

Существуют следующие ранжирующие функции T-SQL: ROW_NUMBER(), NTILE(), RANK(), DENSE_RANK().

ROW_NUMBER() возвращает последовательный номер строки, начиная с 1, в секции результирующего набора независимо от того, есть ли в строках повторяющиеся значения или нет, и имеет следующий синтаксис:

```
ROW_NUMBER()
OVER ( [PARTITION BY <столбцы> ] ORDER BY <столбцы> )
```

Вычислим последовательные номера строк в рамках указанной секции и в соответствии с заданным упорядочиванием (результаты выполнения этого запроса показаны в таблице 4.4):

```
SELECT
   ID, Category, Product, Price, order_date,
   ROW_NUMBER() OVER(PARTITION BY order_date ORDER BY Price)
AS 'row_num'
  FROM Products
```

Таблица 4.4 – Результаты использования инструкции ROW_NUMBER()

ID	category	product	price	order_date	row_num
2	Мышь	Xiaomi Mi Wireless	19.99	2023-05-10	1
1	Мышь	Logitech G Pro	39.99	2023-05-10	2
3	Мышь	FANTECH X9 Thor	25.55	2023-05-12	1
5	Клавиатура	Delux T9	29.99	2023-05-12	2
4	Клавиатура	Redragon K552	34.99	2023-05-12	3

NTILE() распределяет строки упорядоченной секции в заданное количество групп примерно равных размеров на основании переданного числа секций и столбца, по которому проводится упорядочивание. Для каждой строки функция NTILE возвращает номер группы, которой принадлежит строка. Синтаксис функции имеет следующий вид:

```
NTILE ( <количество групп> )  OVER ( [ PARTITION BY <столбцы> ]
ORDER BY <столбцы> )
```

В запросе, приведенном ниже, вычислены одновременно и порядковые номера строк, и номера групп на основе одного и того же критерия упорядочивания (результаты выполнения этого запроса показаны в таблице 4.5).

```
SELECT
   ID, Category, Product, Price, order_date,
   ROW_NUMBER() OVER(ORDER BY Price) AS 'row_num',
   NTILE(2) OVER(ORDER BY Price) AS 'tile'
FROM Products
```

Таблица 4.5 – Результаты использования инструкции NTILE()

ID	category	product	price	order_date	row_num	tile
2	Мышь	Xiaomi Mi Wireless	19.99	2023-05-10	1	1
3	Мышь	FANTECH X9 Thor	25.55	2023-05-12	2	1
5	Клавиатура	Delux T9	29.99	2023-05-12	3	1
4	Клавиатура	Redragon K552	34.99	2023-05-12	4	2
1	Мышь	Logitech G Pro	39.99	2023-05-10	5	2

RANK() возвращает ранг каждой строки в секции результирующего набора. Ранг строки вычисляется как единица плюс количество рангов, находящихся до этой строки. Если есть повторяющиеся значения, функция возвращает одинаковый ранг для таких строчек, пропуская следующий числовой ранг. В отличие от функции ROW_NUMBER, RANK назначает одинаковое значение строкам, претендующим на один ранг. Синтаксис функции имеет следующий вид:

```
RANK()  OVER ( [ PARTITION BY <столбцы> ]
ORDER BY <столбцы> )
```

Далее приведены пример применения функции RANK() и результаты ее использования (таблица 4.6).

```
SELECT
   ID, Category, Product, Price, order_date,
   ROW_NUMBER() OVER(PARTITION BY order_date ORDER BY Price)
AS 'row_num',
   RANK()  OVER(PARTITION BY order_date ORDER BY Price) AS 'rank'
FROM Products
```

Таблица 4.6 – Результаты использования инструкции RANK()

ID	category	product	price	order_date	row_num	rank
2	Мышь	Xiaomi Mi Wireless	19.99	2023-05-10	1	1
1	Мышь	Logitech G Pro	39.99	2023-05-10	2	2
3	Мышь	FANTECH X9 Thor	25.55	2023-05-12	1	1
5	Клавиатура	Delux T9	29.99	2023-05-12	2	2
4	Клавиатура	Redragon K552	34.99	2023-05-12	3	3

DENSE_RANK – возвращает ранг каждой строки в секции результирующего набора без промежутков в значениях ранжирования. Ранг определенной строки равен количеству различных значений рангов, предшествующих строке, увеличенному на единицу. Синтаксис функции имеет следующий вид:

```
DENSE_RANK()   OVER ( [ PARTITION BY <столбцы> ]
ORDER BY <столбцы> )
```

Функции смещения позволяют, перемещаясь по данной секции окна таблицы, рассчитать смещение относительно текущей строки. Предложение ORDER BY должно указываться обязательно.

В T-SQL используются следующие функции смещения:

LAG() – обращается к строке, расположенной с заданным смещением перед началом текущей строки;

LEAD() – обращается к строке, расположенной с заданным смещением после начала текущей строки. По умолчанию смещение равно одной строке.

Синтаксис функций имеет следующий вид:

```
LAG | LEAD ( <столбец> [,<смещение>] [ , <значение по умолчанию> ])
OVER ( [PARTITION BY <столбцы> ]
ORDER BY <столбцы> )
```

Параметр «смещение» задает количество строк до строки перед или после текущей строки, из которой необходимо получить значение. Если значение аргумента не указано, то по умолчанию принимается 1.

Параметр «Значение по умолчанию» должен иметь такой же тип данных, как первый параметр, и используется, когда «смещение» находится за пределами секции. Если не задано, то возвращается NULL.

Функции FIRST_VALUE() и LAST_VALUE() возвращают первое или последнее значение из упорядоченного набора значений в пределах данной секции. Синтаксис обеих функций имеет следующий вид:

```
FIRST_VALUE | LAST_VALUE ( <столбец> )
OVER ( [ PARTITION BY <столбцы> ]
ORDER BY <столбцы> )
```

Применение функций смещения продемонстрировано далее в запросе и проиллюстрировано таблицей 4.7:

```
SELECT
  ID, Category, Product, Price, order_date,
  LAG(Date) OVER(PARTITION BY order_date ORDER BY Price) AS 'Lag',
  FIRST_VALUE(Price) OVER(PARTITION BY Category ORDER BY
order_date) AS 'First_Value',
   LAST_VALUE(Price) OVER(PARTITION BY Category ORDER BY
order_date) AS 'Last_Value'
FROM Products
```

Таблица 4.7 – Результаты использования инструкции LAG(), LEAD(), FIRST_VALUE() и LAST_VALUE()

ID	category	product	price	order_date	lag	first_ value	last_ value
5	Клавиатура	Delux T9	29.99	2023-05-12	2023-05-12	29.99	34.99
4	Клавиатура	Redragon K552	34.99	2023-05-12	2023-05-12	29.99	34.99
2	Мышь	Xiaomi Mi Wireless	19.99	2023-05-10	NULL	19.99	39.99
1	Мышь	Logitech G Pro	39.99	2023-05-10	2023-05-10	19.99	39.99
3	Мышь	FANTECH X9 Thor	25.55	2023-05-12	NULL	19.99	25.55

Задание

Для разрабатываемой БД необходимо написать пять запросов с различными оконными функциями.

Содержание отчета: тема и цель работы; прокомментированный SQL-код выполнения задания.

Контрольные вопросы

1 Что такое оконные функции, чем они отличаются от агрегатных функций?
2 На какие категории можно разделить оконные функции в SQL Server?
2 Какие команды T-SQL используются для записи оконных функций?

Список литературы

1 **Агальцов, В. П.** Базы данных [Электронный ресурс]: в 2 т. Т. 2: Распределенные и удаленные базы данных : учебник / В. П. Агальцов. – Москва : ФОРУМ; ИНФРА-М, 2021. – 271 с. – Режим доступа: https://znanium.com/ catalog/document?id=377105]. – Дата доступа: 25.04.2023.

2 **Агальцов, В. П.** Базы данных [Электронный ресурс]: учебник: в 2 кн. Кн. 1: Локальные базы данных / В. П. Агальцов. – Москва: ФОРУМ; ИНФРА-М, 2020. – 352 с.: ил. – Режим доступа: https: // znanium.com / catalog / document?id = 398558. – Дата доступа: 25.04.2023.

3 **Бен-Ган, И.** Microsoft SQL Server 2012. Создание запросов: учебный курс Microsoft: пер. с англ. / И. Бен-Ган, Д. Сарка, Р. Талмейдж. – Москва : Русская редакция, 2015. – 720 с. : ил.

4 **Бондарь, А. Г.** Microsoft SQL Server 2014 / А. Г. Бондарь. – Санкт-Петербург : БХВ-Петербург, 2015. – 592 с. : ил.

5 **Кузин, А. В.** Базы данных: учебное пособие для студентов высших учебных заведений / А. В. Кузин, С. В. Левонисова. – 6-е изд., стер. – Москва : Академия, 2016. – 320 с.

6 **Куликов, С. С.** Реляционные базы данных в примерах: практическое по-

собие для программистов и тестировщиков [Электронный ресурс] / С. С. Куликов. – Минск : Четыре четверти, 2023. – 424 с. – Режим доступа: https: // svyatoslav.biz / relational_databases_book /. – Дата доступа: 25.04.2023.

7 **Куликов, С. С.** Работа с MySQL, MS SQL Server и Oracle в примерах [Электронный ресурс]: практическое пособие / С. С. Куликов. – Минск : БОФФ, 2022. – 592 с. – Режим доступа: http://svyatoslav.biz/database_book/. – Дата доступа: 25.04.2023.

8 **Мартишин, С. А.** Базы данных. Практическое применение СУБД SQL и NoSQL-типа для проектирования информационных систем [Электронный ресурс]: учебное пособие / С. А. Мартишин, В. Л. Симонов, М. В. Храпченко. – Москва : ФОРУМ ; ИНФРА-М, 2022. – 368 с. – Режим доступа: https: // znanium.com / catalog / product / 1873270. – Дата доступа: 25.04.2023.

9 **Полищук, Ю. В.** Базы данных и их безопасность [Электронный ресурс]: учебное пособие / Ю. В. Полищук, А. С. Боровский. – Москва : ИНФРА-М, 2020. – 210 с. – Режим доступа: https: // znanium.com / catalog / product / 1011088. – Дата доступа: 25.04.2023.

10 **Тарасов, С. В.** СУБД для программиста: базы данных изнутри [Электронный ресурс] / С. В. Тарасов. – Москва : СОЛОН-Пресс, 2020. – 320 с. – Режим доступа: https: // znanium.com / catalog / product / 1227737. – Дата доступа: 25.04.2023.

11 **Шустова, Л. И.** Базы данных [Электронный ресурс]: учебник / Л. И. Шустова, О. В. Тараканов. – Москва: ИНФРА-М, 2017. – 304 с. – Режим доступа: https: // znanium.com / catalog / document?id = 13826. – Дата доступа: 25.04.2023.