

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Астраханский государственный университет имени В. Н. Татищева»
(Астраханский государственный университет им. В. Н. Татищева)

СОГЛАСОВАНО
Руководитель ОПОП

_____ М.В. Коломина

«8» ноября 2022 г.

УТВЕРЖДАЮ
Зав. кафедрой ПМИ

_____ М.В. Коломина

«8» ноября 2022 г.

РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ

«Базы данных»

Составители

**Рахманина А.А., ст. преподаватель, АГУ
Маятин А.В., к.п.н., доцент, доцент ФИТиП, ИТМО**

Направление подготовки /
специальность

**01.03.02 Прикладная математика и
информатика**

Направленность (профиль) ОПОП

**Программирование и искусственный
интеллект**

Квалификация (степень)

бакалавр

Форма обучения

очная

Год приёма

2023

Курс

4

Семестр(ы)

7

1. ЦЕЛИ И ЗАДАЧИ ОСВОЕНИЯ ДИСЦИПЛИНЫ (МОДУЛЯ)

1.1. Целью освоения дисциплины «Базы данных» является освоение способов и возможностей создания и использования баз данных и систем управления базами данных в профессиональной деятельности.

1.2. Задачи освоения дисциплины:

- изучение базы данных как компонент информационной системы, моделей данных и систем управления базами данных,
- изучение методов повышения производительности, надежности и безопасности компонентов хранения данных,
- изучение подходов к построению распределенных баз данных, нереляционных баз данных и областей их эффективного применения
- формирование практических навыков.

2. МЕСТО ДИСЦИПЛИНЫ В СТРУКТУРЕ ОПОП

2.1. Учебная дисциплина «Базы данных» относится к обязательной части и осваивается в 7 семестре.

2.2. Для изучения данной учебной дисциплины (модуля) необходимы следующие знания, умения, навыки, формируемые предшествующими учебными дисциплинами (модулями):

- Дискретная математика.
Знания: теория множеств (определения, свойства множеств, операции над множествами).
Умения: применение основных операций над множествами.
Навыки: представление множеств в компьютерных программах.

2.3. Последующие учебные дисциплины и (или) практики, для которых необходимы знания, умения и навыки, формируемые данной учебной дисциплиной:

- Производственная практика.
- ВКР

3. ПЛАНИРУЕМЫЕ РЕЗУЛЬТАТЫ ОБУЧЕНИЯ ПО ДИСЦИПЛИНЕ

Процесс освоения дисциплины направлен на формирование элементов следующих компетенций в соответствии с ФГОС ВО и ОПОП ВО по данному направлению подготовки / специальности:

- а) общепрофессиональных (ОПК);
 - ОПК-4. Способен понимать принципы работы современных информационных технологий и использовать их для решения задач профессиональной деятельности.
- б) профессиональных (ПК).
 - ПК-6. Способен выполнять работы по созданию и сопровождению информационных систем.
 - ПК-8. Способность понимать, совершенствовать и применять современный математический аппарат.

Таблица 1 – Декомпозиция результатов обучения

Код и наименование компетенции	Планируемые результаты обучения по дисциплине (модулю)		
	Знать (1)	Уметь (2)	Владеть (3)
ОПК-4. Способен понимать принципы работы современных информационных технологий и использовать их для решения задач профессиональной деятельности.	ИОПК-4.1. Современные информационные технологии и программные средства для решения про-	ИОПК-4.2. Использует принципы информационной безопасности при работе с информацией в процессе решения задач профессиональной деятельности	ИОПК-4.3.1. Навыками обоснования принятых решений, планирования и разработки текущих и перспективных планов развития проектов,

	фессиональных задач		оценки эффективности результатов профессиональной деятельности.
ПК-6. Способен выполнять работы по созданию и сопровождению информационных систем.	ИПК-6.1.1. Языков программирования.	ИПК-6.2.1. Кодирования на языках программирования, осуществление установки и настройки системного и прикладного ПО.	ИОПК-6.3.1. Навыками создания и сопровождения информационных систем, поддержка локально развернутых баз данных, поиск и устранение ошибок в работе базы данных.
ПК-8. Способность понимать, совершенствовать и применять современный математический аппарат.	ИПК-8.1.1. Современный математический аппарат	ИПК-8.2.1. Владения методами функционального анализа для решения сложных задач информатики	ИОПК-8.3.1. Применения современного математического аппарата.

4. СТРУКТУРА И СОДЕРЖАНИЕ ДИСЦИПЛИНЫ

Объем дисциплины составляет 5 зачетных единиц, в том числе 60 часов, выделенных на контактную работу обучающихся с преподавателем (из них 30 часов – лекции, 30 часов – лабораторные работы) и 120 часов – на самостоятельную работу обучающихся.

Таблица 2 – Структура и содержание дисциплины

Раздел, тема дисциплины	Семестр	Контактная работа (в часах)			Самост. работа		Форма текущего контроля успеваемости, форма промежуточной аттестации [по семестрам]
		Л	ПЗ	ЛР	КР	СР	
Введение	7	15		15		60	Лабораторные работы
Реализация систем управления базами данных	7	15		15		60	Лабораторные работы
Итого		30		30		120	Экзамен

Таблица 3 – Матрица соотношения разделов, тем учебной дисциплины (модуля) и формируемых компетенций

Раздел, тема дисциплины (модуля)	Кол-во часов	Код компетенции			Общее количество компетенций
		ОПК-4	ПК-6	ПК-8	
Введение	90	+	+	+	3
Реализация систем управления базами данных	90	+	+	+	3
Итого	180				3

Краткое содержание каждой темы дисциплины

№ раздела	Наименование раздела дисциплины	Содержание
1	Введение	Операторы, База данных Деканат, Первое знакомство с СУБД, Основы баз данных, Реляционная модель данных, Нормализация, Реляционное исчисление, Модели и нормализация БД Деканат, Дополнительные отношения БД Деканат, Модели БД, Реляционная алгебра
2	Реализация систем управления базами данных	Обработка запросов, Запросы к БД Деканат, Сложные запросы к БД Деканат, Одновременное обновление БД Деканат, Распределенные базы данных и их задачи, Хранение данных, Процедуры и функции, Транзакции, Типы БД

5. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ПРЕПОДАВАНИЮ И ОСВОЕНИЮ ДИСЦИПЛИНЫ

5.1. Указания для преподавателей по организации и проведению учебных занятий по дисциплине (модулю)

Лекционные занятия

Основной формой реализации теоретического обучения является лекция, которая представляет собой систематическое, последовательное изложение преподавателем-лектором учебного материала теоретического характера. Цель лекции – организация целенаправленной познавательной деятельности студентов по овладению программным материалом учебной дисциплины.

Порядок подготовки лекционного занятия включает в себя выполнение следующих этапов:

- изучение требований программы дисциплины;
- определение целей и задач лекции;
- разработка плана проведения лекции;
- подбор литературы (ознакомление с методической литературой, публикациями периодической печати по теме лекционного занятия);
- отбор необходимого и достаточного по содержанию учебного материала;
- определение методов, приемов и средств поддержания интереса, внимания, стимулирования творческого мышления студентов;
- написание конспекта лекции.

Лекция должна включать следующие разделы:

- формулировку темы лекции;
- указание основных изучаемых разделов или вопросов и предполагаемых затрат времени на их изложение;
- изложение вводной части;
- изложение основной части лекции;
- краткие выводы по каждому из вопросов;
- заключение;
- рекомендации литературных источников по излагаемым вопросам.

Лабораторные занятия

Лабораторное занятие – целенаправленная форма организации педагогического процесса, направленная на углубление научно-теоретических знаний и овладение определенными методами работы, в процессе которых вырабатываются умения и навыки выполнения тех или иных учебных действий в данной сфере науки. Они развивают научное мышление и речь, позволяют проверить знания студентов и выступают как средства оперативной обратной связи.

Правильно организованные лабораторные занятия ориентированы на решение следующих задач:

- обобщение, систематизация, углубление, закрепление полученных на лекциях и в процессе самостоятельной работы теоретических знаний по дисциплине (предмету);
- формирование практических умений и навыков, необходимых в будущей профессиональной деятельности, реализация единства интеллектуальной и практической деятельности;
- выработка при решении поставленных задач таких профессионально значимых качеств, как самостоятельность, ответственность, точность, творческая инициатива.

Состав заданий для лабораторного занятия должен быть спланирован с расчетом, чтобы за отведенное время они могли быть качественно выполнены большинством учащихся.

Лабораторные занятия должны так быть организованы, чтобы студенты ощущали нарастание сложности выполнения заданий, испытывали бы положительные эмоции от переживания собственного успеха в учении, поисками правильных и точных решений.

Самостоятельная работа

Самостоятельная работа – это вид учебной деятельности, которую студент совершает в установленное время и в установленном объеме индивидуально или в группе, без непосредственной помощи преподавателя (но при его контроле), руководствуясь сформированными ранее представлениями о порядке и правильности выполнения действий.

В учебном процессе образовательного учреждения выделяются два вида самостоятельной работы:

- аудиторная – выполняется на учебных занятиях, под непосредственным руководством преподавателя и по его заданию (выполнение самостоятельных работ; выполнение контрольных и практических работ; решение задач);
- внеаудиторная – выполняется по заданию преподавателя, но без его непосредственного участия (подготовка к аудиторным занятиям; изучение учебного материала, вынесенного на самостоятельную проработку; выполнение домашних заданий разнообразного характера; выполнение индивидуальных заданий, направленных на развитие у студентов самостоятельности и инициативы; подготовка к контрольной работе). Внеаудиторные самостоятельные работы представляют собой логическое продолжение аудиторных занятий, проводятся по заданию преподавателя, который инструктирует студентов и устанавливает сроки выполнения задания.

5.2. Указания для обучающихся по освоению дисциплины (модулю)

Лекция

- Лекция – основной вид обучения в вузе.
- В лекции излагаются основные положения теории, ее понятия и законы, приводятся факты, показывающие связь теории с практикой.
- Накануне лекции необходимо повторить содержание предыдущей лекции (а также теорию по изучаемой теме в школьных учебниках геометрии, если эта тема была представлена в них), а затем посмотреть тему очередной лекции по программе (по плану лекций).
- Полезно вести записи (конспекты) лекций: для непонятных вопросов оставлять место при работе над темой лекции с учебными пособиями.
- Записи лекций следует вести в отдельной тетради, оставляя место для дополнений во время самостоятельной работы.
- При конспектировании лекций выделяйте главы и разделы, параграфы, подчеркивайте основное.

Лабораторное занятие

- Лабораторное занятие – наиболее активный вид учебных занятий в вузе. Он предполагает самостоятельную работу над лекциями и учебными пособиями.
- К каждому лабораторному занятию нужно готовиться. Подготовку следует начинать с повторения теории (по записям лекций или по учебному пособию). После этого нужно решать задачи из предложенного домашнего задания.

Организация самостоятельной работы

Самостоятельность в учебной работе способствует развитию заинтересованности студента в изучаемом материале, вырабатывает у него умение и потребность самостоятельно получать знания, что весьма важно для специалиста с высшим образованием. Самостоятельная работа студентов представлена в следующих формах:

- работа с учебной литературой и конспектом лекций с целью подготовки к лабораторным занятиям, составление конспектов тем, выносимых на самостоятельную проработку;
- систематическое выполнение домашних работ.

Таблица 4 – Содержание самостоятельной работы обучающихся

Вопросы, выносимые на самостоятельное изучение	Кол-во часов	Форма работы
--	--------------	--------------

Вопросы, выносимые на самостоятельное изучение	Кол-во часов	Форма работы
Введение	60	Изучение теоретического материала, подготовка в лабораторным работам
Реализация систем управления базами данных	60	Изучение теоретического материала, подготовка в лабораторным работам

5.3. Виды и формы письменных работ, предусмотренных при освоении дисциплины (модуля), выполняемые обучающимися самостоятельно

Дисциплиной «Базы данных» письменные работы не предусмотрены.

6. ОБРАЗОВАТЕЛЬНЫЕ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

При реализации различных видов учебной работы по дисциплине «Базы данных» могут использоваться электронное обучение и дистанционные образовательные технологии.

6.1. Образовательные технологии

Учебные занятия по дисциплине могут проводиться с применением информационно-телекоммуникационных сетей при опосредованном (на расстоянии) интерактивном взаимодействии обучающихся и преподавателя в режимах on-line или off-line в формах.

Таблица 5 – Образовательные технологии, используемые при реализации учебных занятий

Раздел, тема дисциплины (модуля)	Форма учебного занятия		
	Лекция	Практическое занятие, семинар	Лабораторная работа
Введение	Лекция-диалог	Фронтальный опрос, выполнение практических заданий, тематические дискуссии	Фронтальный опрос, выполнение лабораторных заданий, тематические дискуссии
Реализация систем управления базами данных	Лекция-диалог	Не предусмотрено	Фронтальный опрос, выполнение лабораторных заданий, тематические дискуссии

6.2. Информационные технологии

При реализации различных видов учебной и внеучебной работы используются следующие информационные технологии:

- использование возможностей интернета в учебном процессе (использование сайта преподавателя (рассылка заданий, предоставление выполненных работ, ответы на вопросы, ознакомление обучающихся с оценками и т. д.));
- использование электронных учебников и различных сайтов (например, электронных библиотек, журналов и т. д.) как источников информации;
- использование возможностей электронной почты преподавателя;
- использование средств представления учебной информации (электронных учебных пособий и практикумов, применение новых технологий для проведения очных (традиционных) лекций и семинаров с использованием презентаций и т. д.);
- использование интегрированных образовательных сред, где главной составляющей являются не только применяемые технологии, но и содержательная часть, т. е. информационные ресурсы (доступ к мировым информационным ресурсам, на базе которых строится учебный процесс);
- использование виртуальной обучающей среды (LMS Moodle «Электронное образование») или иных информационных систем, сервисов и мессенджеров.

6.3. Современные профессиональные базы данных и информационные справочные системы

6.3.1. Программное обеспечение

Наименование программного обеспечения	Назначение
Adobe Reader	Программа для просмотра электронных документов
Платформа дистанционного обучения LMS Moodle	Виртуальная обучающая среда

Microsoft Office 2013, Microsoft Office Project 2013, Microsoft Office Visio 2013	Пакет офисных программ
7-zip	Архиватор
Microsoft Windows 7 Professional	Операционная система
Kaspersky Endpoint Security	Средство антивирусной защиты
Google Chrome	Браузер
OpenOffice	Пакет офисных программ

6.3.2. Современные профессиональные базы данных и информационные справочные системы

1. Электронная библиотека «Астраханский государственный университет» собственной генерации на платформе ЭБС «Электронный Читальный зал – БиблиоТех». <https://biblio.asu.edu.ru>
2. Электронно-библиотечная система (ЭБС) ООО «Политехресурс» «Консультант студента». www.studentlibrary.ru.
3. Электронная библиотечная система издательства ЮРАЙТ, раздел «Легендарные книги». www.biblio-online.ru
4. Электронный каталог Научной библиотеки АГУ на базе MARK SQL НПО «Информ-систем». <https://library.asu.edu.ru>

7. ФОНД ОЦЕНОЧНЫХ СРЕДСТВ ДЛЯ ПРОВЕДЕНИЯ ТЕКУЩЕГО КОНТРОЛЯ И ПРОМЕЖУТОЧНОЙ АТТЕСТАЦИИ ПО ДИСЦИПЛИНЕ

7.1. Паспорт фонда оценочных средств

При проведении текущего контроля и промежуточной аттестации по дисциплине «Базы данных» проверяется сформированность у обучающихся компетенций, указанных в разделе 3 настоящей программы. Этапность формирования данных компетенций в процессе освоения образовательной программы определяется последовательным освоением дисциплин и прохождением практик, а в процессе освоения дисциплины – последовательным достижением результатов освоения содержательно связанных между собой разделов, тем.

Таблица 6. Соответствие разделов, тем дисциплины, результатов обучения по дисциплине и оценочных средств

№ п/п	Контролируемые разделы, темы дисциплины (модуля)	Код контролируемой компетенции (компетенций)	Наименование оценочного средства
1	Введение	ОПК-4, ПК-6, ПК-8	лабораторная работа
2	Реализация систем управления базами данных	ОПК-4, ПК-6, ПК-8	лабораторная работа

7.2. Описание показателей и критериев оценивания компетенций, описание шкал оценивания

Таблица 7 – Показатели оценивания результатов обучения в виде знаний

Шкала оценивания	Критерии оценивания
5 «отлично»	демонстрирует глубокое знание теоретического материала, умение обоснованно излагать свои мысли по обсуждаемым вопросам, способность полно, правильно и аргументированно отвечать на вопросы, приводить примеры
4 «хорошо»	демонстрирует знание теоретического материала, его последовательное изложение, способность приводить примеры, допускает единичные ошибки, исправляемые после замечания преподавателя
3 «удовлетворительно»	демонстрирует неполное, фрагментарное знание теоретического материала, требующее наводящих вопросов преподавателя, допускает существенные ошибки в его изложении, затрудняется в приведении примеров и формулировке выводов
2 «неудовлетов-»	демонстрирует существенные пробелы в знании теоретического материала, не способен его изложить и ответить на наводящие вопросы преподавателя, не может привести приме-

Шкала оценивания	Критерии оценивания
«отлично»	ры

Таблица 8 – Показатели оценивания результатов обучения в виде умений и владений

Шкала оценивания	Критерии оценивания
5 «отлично»	демонстрирует способность применять знание теоретического материала при выполнении заданий, последовательно и правильно выполняет задания, умеет обоснованно излагать свои мысли и делать необходимые выводы
4 «хорошо»	демонстрирует способность применять знание теоретического материала при выполнении заданий, последовательно и правильно выполняет задания, умеет обоснованно излагать свои мысли и делать необходимые выводы, допускает единичные ошибки, исправляемые после замечания преподавателя
3 «удовлетворительно»	демонстрирует отдельные, несистематизированные навыки, испытывает затруднения и допускает ошибки при выполнении заданий, выполняет задание по подсказке преподавателя, затрудняется в формулировке выводов
2 «неудовлетворительно»	не способен правильно выполнить задания

7.3. Контрольные задания и иные материалы, необходимые для оценки результатов обучения по дисциплине (модулю).

Контроль успеваемости по дисциплине осуществляется с помощью следующих оценочных средств:

ПРОМЕЖУТОЧНАЯ АТТЕСТАЦИЯ

Промежуточная аттестация по дисциплине осуществляется с помощью следующих оценочных средств:

Лабораторная работа 1

Цель работы: Изучение базового синтаксиса инструкции SELECT, выражения WHERE и ORDER BY.

Базовый синтаксис инструкции SELECT

Общий синтаксис инструкции SELECT весьма сложен, однако достаточное представление дает следующее описание:

```
SELECT лист выборки [ INTO новая таблица ]
[ FROM источник данных ] [ WHERE условия поиска ]
[ GROUP BY выражение для группировки ]
[ HAVING условия поиска по группам ]
[ ORDER BY выражение для сортировки [ ASC | DESC ] ]
```

В данном описании надо учитывать, что курсивом указаны пользовательские, обязательные, параметры синтаксиса конструкции на языке T-SQL, а в квадратных скобках, [], – необязательные элементы синтаксиса. Не надо путать использование квадратных скобок [] в описании синтаксиса и в указании идентификаторов. Фигурные скобки, {}, говорят о том, что пользователь обязан выбрать один из вариантов выражений, перечисленных в скобках через символ «вертикальная черта», |. Традиционно все ключевые слова T-SQL пишутся в верхнем регистре, т.е. прописными буквами, но это правило не строгое, язык T-SQL – регистронезависимый.

Инструкция следующего вида вполне допустима:

```
SELECT 'мама мыла раму'
```

В результате выполнения данной инструкции пользователь получит таблицу (результат выполнения инструкции SELECT всегда является таблицей), состоящую из одной строки и

одного столбца, при этом столбец не будет иметь имени.

Следующее изменение инструкции позволит получить таблицу с именованным столбцом:

```
SELECT 'мама мыла раму' AS [просто текст]
```

В данном случае ключевое слово AS говорит о том, что далее будет следовать псевдоним для столбца. Псевдонимы могут быть у столбцов, таблиц и результатов выполнения запросов. Псевдоним, в данном примере *[просто текст]*, всегда должен подчиняться правилу идентификаторов (начинаться с буквы, возможно, использовать цифры, все символы без пробелов), если же по каким-то причинам правило идентификаторов нарушается, то необходимо использовать квадратные скобки. Существует общая рекомендация повсеместного использования квадратных скобок, но на практике их опускают ради повышения скорости набора кода.

качестве источника данных в общем случае выступают таблицы базы данных. Также в качестве источника данных могут выступать отдельные таблицы, результат объединения таблиц, представления, табличные переменные, обобщенные табличные выражения, производные таблицы.

Пример выборки всего содержимого из таблицы:

```
SELECT *
FROM [Production].[Product]
```

Необходимо дать несколько пояснений. Символ «звездочка», *, говорит о том, что из таблицы должны быть выбраны все столбцы. Использовать подобную конструкцию надо с крайней осторожностью, так как она создает значительную нагрузку на сервер и сетевую инфраструктуру. В случае если используется выборка из нескольких таблиц использование «звездочки» может привести к ошибке, так как в результате объединения таблиц может возникнуть ситуация наличия двух и более столбцов с одним именем в результирующем множестве (данный вопрос будет рассмотрен в последующих разделах). Конструкция *[Production].[Product]* представляет собой полное имя таблицы, где имя таблицы – *[Product]*, а *[Production]* – это название схемы. Схема – это поименованное логическое объединение таблиц, используемое для упрощения понимания архитектуры базы данных.

Для получения данных из столбца необходимо указать соответствующее имя столбца, например, запрос, возвращающий все названия продуктов из таблиц *[Product]*:

```
SELECT [Name]
FROM [Production].[Product]
```

Рассмотрим еще один вариант написания запроса:

```
SELECT p.[Name]
FROM [Production].[Product] AS p
```

В данном случае для таблицы *Product* из схемы *Production* введен псевдоним *p*. Разумное использование псевдонимов позволяет увеличить скорость написания кода в MS SQL Management Studio.

Следующая инструкция позволяет получить таблицу с двумя столбцами:

```
SELECT [ProductID], [Name]
FROM [Production].[Product]
```

Рассмотрим запрос:

```
SELECT [ProductID], [Name], [ListPrice]-[StandardCost] AS [discount size]
FROM [Production].[Product]
```

В этом примере пользователь получает таблицу из трех столбцов, данные первых двух столб-

цов взяты непосредственно из таблицы [Product], а третий столбец является результатом выполнения операции вычитания данных одного столбца из данных другого столбца, с использованием псевдонима для упрощения понимания результата.

Еще один вариант:

```
SELECT [ProductID], [Name], 'empty column', GETDATE()
FROM [Production].[Product]
```

Пользователь может также получать не только столбцы данных из таблицы, но и столбец с константным значением (третий столбец), а также столбец, содержащий результат выполнения функции. В данном случае в четвертом столбце выводится текущее время сервера, которое возвращает функция GETDATE():

Обратите внимание на формат даты и времени, которые вернул сервер. Формат зависит от локальных настроек сервера и соответствует одному из нескольких десятков поддерживаемых стандартов.

По устаревшим стандартам требовалось использование ключевого слова ALL:

```
SELECT ALL [Size]
FROM [Production].[Product]
```

Ключевое слово ALL говорит о том, что в результат выборки войдут все возможные значения столбца [Size], эта конструкция в текущей версии T-SQL является конструкцией по умолчанию и на практике всегда опускается.

Получение данных без повторений:

```
SELECT DISTINCT [Size]
FROM [Production].[Product]
```

Ключевое слово DISTINCT определяет, что в выборке будут данные из столбца [Size] без повторений.

```
SELECT DISTINCT [Color], [Size]
FROM [Production].[Product]
```

Этот запрос вернет все существующие в таблице пары значений [Color] и [Size] без повторений.

Выражение ORDER BY

Учебные примеры могут дать ложную иллюзию того, что данные, получаемые в результате выполнения инструкции SELECT, находятся в каком-то упорядоченном виде. Рассмотренный ранее запрос, SELECT [Name] FROM [Production].[Product],

действительно возвращает название всех продуктов в упорядоченном по алфавиту виде, но это всего лишь стечение обстоятельств. В общем случае запрос возвращает данные в том порядке, который определен их физическим расположением в файлах БД и операциями, которые выполняла СУБД для их получения. Однако есть инструкция, позволяющая получить упорядоченный набор данных.

Упорядоченный вывод:

```
SELECT [Name], [ListPrice]
FROM [Production].[Product]
ORDER BY [ListPrice] DESC
```

Инструкция ORDER BY определяет порядок вывода строк, в данном случае будут выведены названия продукта и его цена в виде таблицы, упорядоченной по убыванию цены. Ключевое слово DESC определяет порядок упорядочивания по убыванию, ASC – по возрастанию. Если направление упорядочивания не указано явно, то упорядочивание будет произ-

ведено по возрастанию. Инструкция ORDER BY всегда является завершающей частью запроса.

Упорядочение по нескольким столбцам, с указанием направления для каждого столбца:

```
SELECT [FirstName], [MiddleName], [LastName]
FROM [Person].[Person]
ORDER BY [FirstName] ASC, [MiddleName] DESC, [LastName] ASC
Упорядочение по позиции в выводе, в данном примере по второму столбцу:
SELECT [Name],
[StandardCost] - [ListPrice]
FROM [Production].[Product]
ORDER BY 2
```

Также возможно использование функции для получения параметра упорядочивания:

```
SELECT BusinessEntityID, JobTitle, HireDate
FROM HumanResources.Employee
ORDER BY DATEPART(year, HireDate)
```

Рассмотрим следующий пример:

```
SELECT TOP 3 [Name],
[ListPrice] FROM [Production].[Product] ORDER BY
[ListPrice] DESC
```

Инструкция TOP ограничивает количество строк, которые получит пользователь при выполнении запроса, в примере соответственно будут выведены три продукта и их цены в отсортированном наборе. Необходимо сделать замечание:

возможно, существует некоторый продукт, условно говоря, четвертый, цена которого равна цене третьего продукта, но он не попадет в выборку для пользователя. К сожалению, мы не можем заранее определить, какой из продуктов, третий или четвертый, с равной ценой, в отсортированном списке попадет в пользовательскую выборку при использовании конструкции TOP 3.

Инструкция WITH TIES позволяет выводить все строки, «соперничающие за последнее место»:

```
SELECT TOP 3 WITH TIES [Name], [ListPrice]
FROM [Production].[Product]
ORDER BY [ListPrice] DESC
```

Приведенный пример использования инструкции WITH TIES позволяет вывести на экран не только первые три продукта из набора, отсортированного по убыванию цены, но и все продукты, цена которых равна цене третьего продукта. Данная инструкция работает исключительно с SELECT и только при использовании оператора упорядоченного вывода ORDER BY.

Вывод доли строк из результирующего набора:

```
SELECT TOP 3 PERCENT [Name], [ListPrice]
FROM [Production].[Product]
ORDER BY [ListPrice] DESC
```

В случае использования конструкции TOP *число* PERCENT пользователь получает некоторый процент строк, равный *числу* из результирующего набора. Процент строк округляется до следующего целого.

Использование инструкции INTO позволяет создать новую таблицу и поместить в нее результат выборки:

```
SELECT TOP 3 PERCENT [Name], [ListPrice] INTO Tmp
FROM [Production].[Product]
ORDER BY [ListPrice] DESC
```

Выполнение данного запроса будет разбито на два этапа. На первом этапе будет создана таблица Tmp. На втором этапе в нее будет помещен результат выполнения запроса. Необходимо добавить несколько замечаний. Если по каким-то причинам при выполнении запроса произойдет сбой, то будет создана пустая таблица. Использование инструкции упорядоченного вывода выборки, ORDER BY, не гарантирует сохранения порядка строк в созданной таблице. Если результирующий набор дает набор строк, не обладающий свойством уникальности, то все равно таблица будет создана, данные будут добавлены, но никакие первичные ключи пользователю не будут доступны, хотя на физическом уровне они будут реализованы СУБД. Для использования инструкции INTO необходимо иметь права, аналогичные правам создания обычной таблицы.

Выражение WHERE

Инструкция SELECT не позволяет выбрать отдельные строки из источника, но можно выбрать строки, удовлетворяющие определенным условиям – наличию тех или иных значений в столбцах.

Простой пример использования предложения WHERE:

```
SELECT [Name], [ListPrice]
FROM [Production].[Product]
WHERE [ListPrice]=3578.27
```

В данном случае будут выведены названия продуктов, у которых цена равна 3578.27, а также их цена. Так как столбец [ListPrice] имеет тип mONey, то использование оператора сравнения на равенство допустимо.

В следующем примере ограничения затрагивают два столбца:

```
SELECT [Name], [ListPrice]
FROM [Production].[Product]
WHERE [ListPrice]=3578.27 AND [Size]=62
```

Пользователь получит название таких товаров, у которых цена равна 3578.27 и размер равен 62, и их цену, для этого был использован оператор AND – логическое И. Существует еще два логических оператора, OR – логическое ИЛИ и NOT – логическое отрицание.

Можно написать достаточно сложный запрос:

```
SELECT [Name] FROM [Production].[Product]
WHERE
([ListPrice]>20 AND [Color]='Red')
OR
([ListPrice]>25 AND [Color]='Black')
```

В данном случае пользователь получит название таких товаров, у которых цена выше 20 и цвет Red, или цена выше 25 и цвет Black.

При работе с числовыми данными бывают ситуации, когда необходимо использовать некоторый диапазон данных. Например, стоит вопрос – найти названия товаров, цена которых лежит в диапазоне от 20 до 40, включая границы диапазона. Реализовать этот запрос можно с помощью логического оператора AND.

```
SELECT [Name] FROM [Production].[Product]
WHERE [ListPrice]>=20 AND [ListPrice]<=40
```

Однако существует специальный оператор BETWEEN, который определяет диапазон для

проверки. Общий синтаксис этого оператора выглядит так:

выражение [NOT] BETWEEN *начало_диапазона* AND *конец_диапазона*

Оператор возвращает значение TRUE, если *выражение* входит, или не входит, если используется оператор NOT, в диапазон от *начало_диапазона* до *конец_диапазона* включительно.

С помощью этого оператора предыдущий запрос можно реализовать следующим образом:

```
SELECT [Name] FROM [Production].[Product]
WHERE [ListPrice] BETWEEN 20 AND 40
```

Если стоит задача сравнения выражения с некоторым набором значений, то могут применяться следующие операторы – IN, ANY (или его синоним – SOME), ALL.

Оператор IN определяет принадлежность значения одному и значений в списке.

```
SELECT [Name] FROM [Production].[Product]
WHERE [Color] IN ('Red', 'Black', 'Silver')
```

Приведенный выше запрос возвращает название товаров, у которых цвет либо Red, либо Black, либо Silver. Стоит обратить внимание, что в скобках может быть не просто набор значений, но и выражение, которое формирует такой набор, например, оператор SELECT.

Операторы ALL и ANY будут рассмотрены позже, в работе с подзапросами.

При работе со строковыми значениями возникает необходимость проверки на соответствие значения не строке, а шаблону, например, требуется найти все продукты, начинающиеся на букву 'D'. Для этого используется оператор LIKE:

```
SELECT [Name] FROM [Production].[Product]
WHERE [Name] LIKE 'D%'
```

При использовании оператора LIKE можно использовать оператор NOT. Необходимо помнить, что существуют параметры сравнения, определяемые настройкой СУБД и отдельных таблиц, но в общем случае СУБД не делает различия между строчными и прописными буквами.

В строковых шаблонах допускаются следующие символы:

% – символ-шаблон, заменяющий любую последовательность символов;

_ (подчеркивание) – символ-шаблон, заменяющий любой одиночный символ;

[] – заменяет одиночный символ, указанный в угловых скобках, можно перечислить символы, или диапазон (через дефис) символов;

[^] – заменяет одиночный символ, не указанный в угловых скобках, можно перечислить символы, или диапазон (через дефис) символов.

Допускается использование ESCAPE последовательностей:

```
WHERE СТРОКА LIKE '%[a-f][^xyz]_30!%%' ESCAPE '!'
```

Данный пример является частью запроса, который, в частности, проверяет, совпадают ли значения в столбце СТРОКА со следующим шаблоном – любое количество символов, далее один из символов диапазона от 'a' до 'f' включительно, далее любой символ, кроме символов 'x', 'y' или 'z', далее еще один любой символ, далее 30%, и опять последовательность любых символов. Символ '!' является эскейп-символом и говорит о том, что следующий за ним символ, в данном случае '%', не надо рассматривать как управляющий; таким образом, в пример включен символ %, который в общем случае является служебным.

В реляционных базах данных существует особое значение – NULL. Это значение не является нулевым значением в математическом понимании нуля, это неопределённость, которая сообщает пользователям, что в данный момент времени значение атрибута не определено. Для работы с этим значением нельзя использовать оператор сравнения на равенство или любой другой оператор сравнения. Проверка на определенность осуществляется с помощью опе-

ратора IS.

```
SELECT [Name] FROM [Production].[Product]
WHERE [Color] IS NULL
```

Данный запрос возвращает название товаров, для которых цвет не определен.

```
SELECT [Name] FROM [Production].[Product]
WHERE [Size] IS NOT NULL
```

Этот запрос возвращает название товаров, у которых определен размер, т.е. он не является NULL.

Примеры запросов с решениями

Описание функций и архитектура приведены в приложении.

1. Получить все названия товаров из таблицы Product.

```
SELECT p.Name
FROM [Production].[Product] AS p
```

В данном случае для таблицы Product из схемы Production введен псевдоним p. Разумное использование псевдонимов позволяет увеличить скорость написания кода в MS SQL Management Studio.

2. Получить все названия товаров в системе, цена которых (listprice) выше 200.

```
SELECT p.Name
FROM [Production].[Product] AS p
WHERE p.ListPrice>200
```

3. Получить все названия товаров в системе цена которых (listprice) выше 200 и у которых первая буква в названии “S”.

```
SELECT p.Name
FROM [Production].[Product] AS p
WHERE p.ListPrice>200 AND p.Name like 's%'
```

4. Получить все названия товаров в системе, цена которых (listprice) выше 200 и у которых в названии есть сочетание символов “are”.

```
SELECT p.Name
FROM [Production].[Product] AS p
WHERE p.ListPrice>200 AND p.Name like '%are%'
```

5. Получить все названия товаров в системе, в названии которых третий символ – либо буква “s”, либо буква “r”. Решить задачу как минимум двумя способами.

```
SELECT p.Name
FROM [Production].[Product] AS p
WHERE p.Name like '__s%' or p.Name like '__r%'
```

или

```
SELECT p.Name
FROM [Production].[Product] AS p
WHERE p.Name like '__[sr]%'
```

7. Получить все названия товаров в системе, в названии которых ровно 5 символов.

```
SELECT p.Name
FROM [Production].[Product] AS
p WHERE p.Name like '_____'
```

Или второй вариант

```
SELECT p.Name
FROM [Production].[Product] AS p
WHERE len(p.Name)=5
```

Во втором примере используется строковая функция len. Строковые функции даны в приложении.

8. Найти, без повторов, номера товаров, которые были куплены хотя бы один раз.

```
SELECT DISTINCT sod.ProductID
FROM [Sales].[SalesORDERDetail] AS sod
```

9. Найти список всех возможных стилей (style) продукта, без повторов.

```
SELECT DISTINCT p.Style
FROM [Production].[Product] AS p
WHERE p.Style is NOT null
```

10. Написать запрос, который возвращает названия товаров, которые были произведены между мартом 2011 года и мартом 2012 года включительно (необходимо учитывать формат даты)

```
SELECT p.Name
FROM [Production].[Product] AS p
WHERE p.SellStartDate>='2011-01-03' AND p.SellStartDate<='2012-31-03'
```

11. Найти максимальную стоимость товара (отпускная цена ListPrice) из тех, которые были произведены, начиная с марта 2011 года.

```
SELECT max(p.ListPrice) AS [max price]
FROM [Production].[Product] AS p
WHERE p.SellStartDate>='2011-01-03'
```

12. Вывести название и цвет продукта, отсортированные по названию продукта по возрастанию.

```
SELECT p.Name, p.Color
FROM [Production].[Product] AS p
ORDER BY p.Name ASC
```

13. Вывести названия продукта, цвет и отпускную цену для таких товаров, у которых цвет определен и цена отлична от нуля, и отсортировать полученный список по возрастанию цвета товара и убыванию отпускной цены.

```
SELECT p.Name, p.Color, p.ListPrice
FROM [Production].[Product] AS p
WHERE p.Color is NOT null AND p.ListPrice!=0
ORDER BY p.Color, p.ListPrice desc
```

14. Получить название продукта и разницу между отпускной стандартной ценой продукта и стандартной ценой продукта для тех товаров, у которых эти показатели не равны нулю.

```
SELECT p.Name, p.ListPrice-p.StandardCost
FROM [Production].[Product] AS p
WHERE p.ListPrice!=0 AND p.StandardCost!=0
```

15. Найти название самого дорогого товара, исходя из предположения, что нет двух товаров с одинаковой ценой.

```
SELECT top 1 with ties p.Name
FROM [Production].[Product] AS
p ORDER BY p.ListPrice desc
```

16. Найти список товаров, произведенных в 2005 году.

```
SELECT p.Name
FROM [Production].[Product] AS p
WHERE datepart(YEAR,p.SellStartDate)=2005
```

Задания для самостоятельной работы

1. Найти и вывести на экран названия продуктов, их цвет и размер.
2. Найти и вывести на экран названия, цвет и размер таких продуктов, у которых цена более 100.
3. Найти и вывести на экран название, цвет и размер таких продуктов, у которых цена менее 100 и цвет Black.
4. Найти и вывести на экран название, цвет и размер таких продуктов, у которых цена менее 100 и цвет Black, упорядочив вывод по возрастанию стоимости продуктов.
5. Найти и вывести на экран название и размер первых трех самых дорогих товаров с цветом Black.
6. Найти и вывести на экран название и цвет таких продуктов, для которых определен и цвет, и размер.
7. Найти и вывести на экран не повторяющиеся цвета продуктов, у которых цена находится в диапазоне от 10 до 50 включительно.
8. Найти и вывести на экран все цвета таких продуктов, у которых в имени первая буква 'L' и третья 'N'.
9. Найти и вывести на экран названия таких продуктов, которых начинаются либо на букву 'D', либо на букву 'M', и при этом длина имени – более трех символов.
10. Вывести на экран названия продуктов, у которых дата начала продаж – не позднее 2012 года.
11. Найти и вывести на экран названия всех подкатегорий товаров.
12. Найти и вывести на экран названия всех категорий товаров.
13. Найти и вывести на экран имена всех клиентов из таблицы Person, у которых обращение (Title) указано как «Mr.».
14. Найти и вывести на экран имена всех клиентов из таблицы Person, для которых не определено обращение (Title).

Лабораторная работа 2

Цель работы: Изучение выражения GROUP BY – группировка данных

Одной из ключевых задач СУБД является выполнение операций, связанных с обработкой данных. Это необходимо для анализа информации, а также для формирования сложных запросов. Выражение GROUP BY позволяет разделить результаты выполнения оператора SELECT на группы по признаку, с целью выполнения статистических операций над группами.

Простой запрос:

```
SELECT [Name], [ListPrice], [Color] FROM [Production].[Product]
```



```
WHERE [Color] IS NOT NULL
```

В результате выполнения этого запроса пользователь получит название, стоимость и цвет таких товаров, у которых цвет определен. Это полезная и важная информация, однако у пользователя может появиться более сложная задача, например, посчитать количество товаров того или иного цвета.

Существует набор функций, которые позволяют выполнить операцию над набором значений в столбце. Такие функции называются агрегирующими:

SUM ([ALL | DISTINCT] expressiON) – сумма значений в столбце;

MAX ([ALL | DISTINCT] expressiON) – максимальное значение в столбце; MIN ([ALL | DISTINCT] expressiON) – минимальное значение в столбце; AVG ([ALL | DISTINCT] expressiON) – среднее значение в столбце;

COUNT ({ [[ALL | DISTINCT] expressiON] | * }) – количество строк в столбце, с учетом NULL значений.

Приведенный общий синтаксис функций рассмотрим на примере: SELECT COUNT(DISTINCT [Color]) FROM [Production].[Product] WHERE [Color] IS NOT NULL

В данном случае будет выполнен SELECT, который отберет из таблицы все строки, для которых цвет определен, и будет выполнена агрегирующая функция COUNT, которая посчитает количество различных цветов. Инструкция DISTINCT обеспечивает выбор неповторяющихся значений цвета, а функция COUNT считает количество полученных значений.

Запрос следующего вида фактически возвращает количество строк, для которых определен цвет продукта:

```
SELECT COUNT([Color]) FROM [Production].[Product]
WHERE [Color] IS NOT NULL
```

Он полностью эквивалентен следующему запросу:

```
SELECT COUNT(*) FROM [Production].[Product]
WHERE [Color] IS NOT NULL
```

Рассмотрим работу функции MAX:

```
SELECT MAX([ListPrice]) FROM [Production].[Product]
WHERE [Color]='Red'
```

Этот запрос возвращает максимальную цену продукта, у которого цвет 'Red'.

Выражение GROUP BY позволяет разделить результат выполнения запроса на группы и выполнить ту или иную функцию над каждой из групп.

```
SELECT [Color], COUNT(*) AS 'Amount'
FROM [Production].[Product]
WHERE [Color] IS NOT NULL
GROUP BY [Color]
```

В данном запросе все строки, для которых выполнено условие «цвет определен», разделены на группы по цвету, и над каждой группой выполнено функция COUNT. Результатом станет два столбца – цвет и количество строк с данным цветом в общей выборке.

Необходимо сделать несколько важных дополнений. При группировке запрос возвращает только те данные, которые имеют одно и то же значение для всей группы. Запрос вида

```
SELECT [Name], [Color], COUNT(*) AS 'Amount'
FROM [Production].[Product]
WHERE [Color] IS NOT NULL
GROUP BY [Color]
```

не будет выполнен, так как в группе может существовать несколько товаров с разными названиями. Даже если фактически в каждой группе с одним и тем же цветом все товары имеют одно и то же название, все равно это будет ошибкой. Так, при группировке нельзя использовать псевдонимы столбцов как признак группировки, нельзя группировать по столбцам с типами данных text, ntext или image, но можно группировать по результату выполнения функции над этими столбцами, например функции приведения типа или строковой функции. Нельзя использовать в качестве признака группировки подзапросы и данные типа XML. Нельзя использовать в качестве признака группировки столбец индексированного представления.

Если столбец, по которому осуществляется группировка, содержит значения NULL, то они будут рассмотрены как идентичные и будут образовывать отдельную группу.

Допускается группировка по результату выполнения простых математических операций, а также группировка по двум и более столбцам:

```
SELECT Столбец1, Столбец2 FROM Таблица GROUP BY Столбец1, Столбец2;
SELECT Столбец1 + Столбец2 FROM Таблица GROUP BY Столбец1, Столбец2;
SELECT Столбец1 + Столбец2 FROM Таблица GROUP BY Столбец1 + Столбец2;
```

```
SELECT Столбец1 + Столбец2 + константное_значение FROM Таблица
GROUP BY Столбец1, Столбец2.
```

Рассмотрим следующий пример:

```
SELECT [Color], [Style], COUNT(*) AS 'Amount'
FROM [Production].[Product]
WHERE [Color] IS NOT NULL AND [Style] IS NOT NULL
GROUP BY [Color], [Style]
```

Запрос возвращает количество продуктов с общим цветом и стилем, упомянутых в таблице продукты», с учетом того, что у этих товаров и цвет, и стиль определены.

Выражение GROUP BY используется совместно с выражением HAVING, которое определяет условие поиска группы.

```
SELECT [Color], COUNT(*) AS 'Amount'
FROM [Production].[Product]
WHERE [Color] IS NOT NULL
GROUP BY [Color]
HAVING COUNT(*) > 10
```

Данный запрос выдаст название цветов, количество товаров данного цвета, но только для тех групп, где это количество больше 10. Естественно, будет выполнено условие, что цвет товаров определен. Условие поиска группы может содержать также логические операторы.

Вот пример такой возможности:

```
SELECT [Color], COUNT(*) AS 'Amount'
FROM [Production].[Product]
WHERE [Color] IS NOT NULL
GROUP BY [Color]
HAVING COUNT(*) > 10 AND MAX([ListPrice]) > 3000
```

Стоит отметить, что операция группировки весьма затратна с точки зрения процессорной мощности и времени, неправильная конструкция может существенно увеличить время выполнения запроса. Например, стоит задача, рассмотренная в начале работы: посчитать количество товаров каждого цвета,

исключив товары, для которых цвет не определен. Выше эта задача была решена, однако формально допустимо следующее решение:

```
SELECT [Color], COUNT(*) AS 'Amount'
FROM [Production].[Product]
GROUP BY [Color]
HAVING [Color] IS NOT NULL
```

Такой запрос будет выполняться дольше, так как сначала сформируются все группы, и только потом будет применено условие для выбора группы, т.е. будет отброшена группа, для которой цвет не определён. Время на формирование группы с неопределённым цветом фактически будет потрачено впустую.

Использование GROUP BY допускает использование ORDER BY, TOP.

Существуют специальные конструкции, выполняющие группировку с определёнными правилами. Рассмотрим конструкцию GROUP BY ROLLUP, которая выводит и общие и промежуточные итоги группировки. Частный синтаксис будет такой:

```
SELECT столбец1, столбец2, столбец3, COUNT(*)
FROM таблица
GROUP BY ROLLUP (столбец1, столбец2, столбец3)
```

В данном случае будут выведены итоги общей группировки по трем столбцам, а также добавлены результаты промежуточной группировки. В данном примере промежуточная группировка это строки полученные в результате группировки по первым двум столбцам, третий столбец будет иметь значение NULL, это строки полученные в результате группировки по первому столбцу, второй и третий будут иметь значение NULL, и одна строка где первый, второй и третий столбцы будут иметь значение NULL.

Для понимания ее работы нам понадобится простой пример группировки.

```
SELECT [Color], [Style], [Class], COUNT(*)
FROM [Production].[Product]
GROUP BY [Color], [Style], [Class]
```

В данном случае пользователь увидит окончательный результат работы группировки по трем столбцам, четвертый столбец это число продуктов, в группе с одинаковым классом, стилем и цветом.

Если выполнить запрос с использованием GROUP BY ROLLUP:

```
SELECT [Color], [Style], [Class], COUNT(*)
FROM [Production].[Product]
GROUP BY ROLLUP ([Color], [Style], [Class])
```

то пользователь получит все те же строки, как и при обычном GROUP BY, а также дополнительный набор с группировкой по цвету и стилю, дополнительный набор с группировкой только по цвету и строку без группировки.

Схожим образом работает операция GROUP BY CUBE (), только к базовой группировке GROUP BY добавляются все возможные сочетания параметров группировки. Например, группировка GROUP BY CUBE (столбец1, столбец2), даст все возможные сочетания группировок, базовую группировку (столбец1, столбец2), и возможные сочетания (столбец1, NULL), (NULL, столбец2), (NULL, NULL).

Конструкция GROUP BY GROUPING SETS () позволяет объединить несколько операций GROUP BY.

```
SELECT COUNT(*)
FROM [Production].[Product]
```

GROUP BY GROUPING SETS (([Color]), ([Size]))

Данный запрос вернет данные, сгруппированные по цвету, и еще набор строк, где данные сгруппированы по размеру.

Примеры запросов с решениями

1. Найти номера первых трех подкатегорий (ProductSubcategoryID) с наибольшим количеством товаров.

```
SELECT TOP WITH TIES 3 [ProductSubcategoryID]
FROM [Production].[Product]
WHERE [ProductSubcategoryID] IS NOT NULL
GROUP BY [ProductSubcategoryID]
ORDER BY COUNT(*) DESC
```

2. Разбить продукты по количеству символов в названии, для каждой группы определить количество продуктов.

```
SELECT LEN([Name]), COUNT(*)
FROM [Production].[Product]
GROUP BY LEN([Name])
```

3. Проверить, есть ли продукты с одинаковым названием, если есть, то вывести эти названия.

```
SELECT [Name]
FROM [Production].[Product]
GROUP BY [ProductID], [Name]
HAVING COUNT(*) > 1
```

Задания для самостоятельной работы

1. Найти и вывести на экран количество товаров каждого цвета, исключив из поиска товары, цена которых меньше 30.
2. Найти и вывести на экран список, состоящий из цветов товаров, таких, что минимальная цена товара данного цвета более 100.
3. Найти и вывести на экран номера подкатегорий товаров и количество товаров в каждой категории.
4. Найти и вывести на экран номера товаров и количество фактов продаж данного товара (используется таблица SalesORDERDetail).
5. Найти и вывести на экран номера товаров, которые были куплены более пяти раз.
6. Найти и вывести на экран номера покупателей, CustomerID, у которых существует более одного чека, SalesORDERID, с одинаковой датой
7. Найти и вывести на экран все номера чеков, на которые приходится более трех продуктов.
8. Найти и вывести на экран все номера продуктов, которые были куплены более трех раз.
9. Найти и вывести на экран все номера продуктов, которые были куплены или три или пять раз.
10. Найти и вывести на экран все номера подкатегорий, в котором относится более десяти товаров.
11. Найти и вывести на экран номера товаров, которые всегда покупались в одном экземпляре за одну покупку.
12. Найти и вывести на экран номер чека, SalesORDERID, на который приходится с наибольшим разнообразием товаров купленных на этот чек.

13 Найти и вывести на экран номер чека, SalesORDERID с наибольшей суммой покупки, исходя из того, что цена товара – это UnitPrice, а количество конкретного товара в чеке – это ORDERQty.

14 Определить количество товаров в каждой подкатегории, исключая товары, для которых подкатегория не определена, и товары, у которых не определен цвет.

15 Получить список цветов товаров в порядке убывания количества товаров данного цвета

16 Вывести на экран ProductID тех товаров, что всегда покупались в количестве более 1 единицы на один чек, при этом таких покупок было более двух.

Лабораторная работа 3

Цель работы: Выборка данных из нескольких таблиц.

Структурированный язык запросов позволяет получать данные из нескольких таблиц одновременно. Существует простой и понятный синтаксис подобных запросов, но для полного понимания необходимо рассмотреть несколько общих вопросов.

Допустим, пользователь имеет две таблицы. Первая таблица будет называться Корма, и в ней один столбец – название корма. Для примера в таблице будет две строки: сено, мясо. Вторая таблица будет называться Животные и также будет иметь один столбец – название животного, в данном случае будет две строки: собака, лошадь.

Вполне допустим следующий запрос:

```
SELECT [название корма], [название животного] FROM Корма, Животные
```

Однако полученный результат вряд ли устроит пользователя: сено со-

бака

сено лошадь

мясо собака

мясо лошадь

Чтобы результат соответствовал реальному миру, СУБД должно иметь возможность исключить не существующие в реальном мире варианты сочетания корма и животного. Модифицируем наши таблицы. Добавим в таблицу Корма столбец «номер корма», это будет первичный ключ для этой таблицы. В таблицу Животные необходимо будет добавить два столбца. Столбец «номер животного» станет первичным ключом этой таблицы, а еще один столбец, «номер корма», станет внешним ключом, связанным с первичным ключом таблицы Корма. Естественно, значения в столбце «номер корма» таблицы Животные должен содержать значения, которые удовлетворяют двум требованиям. Во-первых, есть точно такие же значения в столбце «номер корма» таблицы Корма. Во-вторых, эти значения соответствуют реальному миру, т.е. в строке, где содержится информация о лошади, должен стоять число, равное номеру сена, а в строке, где упомянута собака, номер ее корма должен соответствовать номеру мяса.

Используемые определения: суперключ, потенциальный ключ, первичный и внешний ключ.

Суперключ – атрибут или множество атрибутов, единственным образом идентифицирующие кортеж.

Потенциальный ключ – суперключ, который не содержит подмножества, также являющегося суперключом данного отношения. Отношение может иметь несколько потенциальных ключей. Если потенциальный ключ состоит из нескольких атрибутов, он называется со-

ставным ключом.

Первичный ключ – потенциальный ключ, который выбран для уникальной идентификации кортежей внутри отношения.

Внешний ключ – атрибут или множество атрибутов внутри отношения, которое соответствует потенциальному ключу некоторого (может быть, того же самого) отношения. Внешние ключи позволяют описать связь между отношениями.

После добавления столбцов и внесения необходимых данных можно выполнить запрос следующего вида:

```
SELECT [название корма], [название животного] FROM Корма, Животные
WHERE Корма.[Номер корма]=Животные.[Номер корма]
```

Он даст результат, который соответствует реальному миру: сено лошадь

мясо собака

В данном случае СУБД получила однозначную инструкцию – не включать в результирующий набор данных, полученный из двух таблиц, те строки, для которых не выполняется равенство внешнего и первичного ключей.

Такой способ соединения таблиц допустим. Пользователь может производить соединения двух и более таблиц, используя сравнения на равенство или неравенство значений столбца одной таблицы со значениями столбца другой таблицы. Надо отметить, что подобные соединения не подразумевают, что столбцы обязательно являются первичным и внешним ключом соответственно.

У подобного соединения есть один существенный недостаток. Операция сравнения на равенство не применима к значениям NULL. Допустим, у нас появилась информация о животном, например, шимпанзе, для которого корм не определен. В итоговую выборку это животное не попадет. Для разрешения подобной трудности используется операция соединения JOIN. Общий синтаксис, стандартный для всех реляционных СУБД выглядит так:

```
FROM      первая_таблица      ТИП_СОЕДИНЕНИЯ      вторая_таблица
        [ON (условия_соединения) ]
```

Порядок следования таблиц при соединении играет важную роль, от этого зависит результат при использовании некоторых типов соединений. Иногда таблицы называют левая и правая таблицы.

Тип соединения определяет принцип соединения таблиц. *Условия соединения* определяют условия сравнения значений в столбцах соединяемых таблиц – равенство или неравенство.

Рассмотрим все типы соединения. Соединение INNER JOIN.

Рассмотрим запрос следующего вида:

```
SELECT p.Name, s.Name
FROM      [Production].[Product]      p      INNER
        JOIN [Production].[ProductSubcategory] s
ON p.ProductSubcategoryID=s.ProductSubcategoryID
```

Пользователь получит список названий товаров и названий категорий, к которым относится товар, при этом не будут рассмотрены товары, для которых подкатегория не определена. INNER JOIN не учитывает NULL значения как в левой, так и в правой таблице при объединении. Синтаксически допустимо и традиционно используется сокращение названия соединения INNER JOIN до просто JOIN.

В рассмотренном выше запросе объединяются таблицы Product и ProductSubcategory, каж-

дая из которых имеет столбец с именем Name. Чтобы различать эти столбцы при объединении, были введены псевдонимы таблиц, и к столбцу Name обращение идет через оператор доступ – точка.

Соединение LEFT JOIN.

LEFT JOIN учитывает значения NULL из левой таблицы. Таким образом нижеприведенный запрос вернет названия продуктов и название подкатегорий, в том числе и те случаи, когда у продукта не определена подкатегория. Если подкатегория продукта не определена, то ее название будет NULL.

```
SELECT p.Name, s.Name
FROM      [Production].[Product]          p          LEFT
          JOIN [Production].[ProductSubcategory] s
ON p.ProductSubcategoryID=s.ProductSubcategoryID
```

Соединение RIGHT JOIN учитывает значения NULL из правой таблицы. Рассмотрим запрос:

```
SELECT p.Name, s.Name
FROM      [Production].[Product]          p          RIGHT
          JOIN [Production].[ProductSubcategory] s
ON p.ProductSubcategoryID=s.ProductSubcategoryID
```

В данном случае соединение RIGHT JOIN учитывает ситуацию, в столбце правой таблицы, ProductSubcategory, есть значения NULL. Это правило формально соблюдено СУБД, несмотря на то, что в правой таблице, ProductSubcategory, столбец для соединения, ProductSubcategoryID, является первичным ключом.

Соединение FULL OUTER JOIN учитывает значение NULL и в левой, и в правой таблице. На практике синтаксическая единица FULL OUTER JOIN сокращается до FULL JOIN.

Рассмотрим пример. Даны две таблицы, tA и tB, со столбцами [tAvalue] и [tBvalue]. Столбец [tAvalue] имеет следующие значения: 1, 2, 3 и NULL. Столбец [tBvalue] имеет значения: 2, 3, 4, и NULL.

Выполнив запрос

```
SELECT [tAvalue], [tBvalue] FROM [dbo].[tA] FULL JOIN [dbo].[tB] ON
tA.[tAvalue]=tB.[tBvalue]
```

пользователь получит следующие пары значений: (1, NULL), (2,3), (3,3), (NULL, NULL), (NULL,4), (NULL, NULL).

В данном случае был рассмотрен пример соединения, в котором условие соединения не связано с первичным или внешним ключом. Также надо учитывать, что запрос вернул две строки со значением NULL в каждом из столбцов.

Соединение CROSS JOIN производит полное декартово произведение двух таблиц и не имеет условия соединения. Рассмотрим запрос к таблицам из предыдущего примера

```
SELECT [tAvalue], [tBvalue] FROM
[dbo].[tA] CROSS JOIN [dbo].[tB]
```

Этот запрос вернет все возможные сочетания значений [tAvalue] и [tBvalue], т.е. шестнадцать строк. Соединение CROSS JOIN может потребовать значительных ресурсов СУБД и серьезно затруднить работу других пользователей.

Соединение таблиц допускает соединение таблицы со своей копией. Стоит задача – найти такие продукты, для которых существуют продукты с таким же названием. Задачу можно решить с использованием выражения GROUP BY, а можно и с использованием самосоединения.

```
SELECT p1.Name
```

```
FROM [Production].[Product] p1 JOIN [Production].[Product] p2
ON p1.Name=p2.Name AND
p1.ProductID!=p2.ProductID
```

Стоит обратить внимание что в данном запросе обязательно использование псевдонимов. Для соединения использованы два условия, одно из которых – на неравенство.

Соединение таблиц допускает использование ключевых слов WHERE, GROUP BY, HAVING, ORDER BY и других. Допускается соединение более чем двух таблиц.

В нижеприведенных примерах используется схема данных из первой лабораторной работы, а также схема, приведенная в приложении к данной работе.

Примеры запросов с решениями

1 Найти название продуктов и название подкатегорий этих продуктов, у которых отпускная цена больше 100, не включая случаи, когда продукт не относится ни к какой подкатегории.

```
SELECT P.Name, PSC.Name
FROM [Production].[Product] AS P INNER JOIN [Production].[ProductSubcategory] AS PSC
ON P.ProductSubcategoryID=PSC.ProductSubcategoryID
WHERE [ListPrice]>100
```

2 Найти название продуктов и название подкатегорий этих продуктов, у которых отпускная цена больше 100, включая случаи, когда продукт не относится ни к какой категории.

```
SELECT P.Name, PSC.Name
FROM [Production].[Product] AS P LEFT JOIN [Production].[ProductSubcategory] AS PSC
ON P.ProductSubcategoryID=PSC.ProductSubcategoryID
WHERE [ListPrice]>100
```

3 Найти название продуктов и название категорий из таблицы ProductCategory, с которой связан этот продукт, не включая случаи, когда у продукта нет подкатегории.

```
SELECT P.Name, PC.Name
FROM [Production].[Product] AS P INNER JOIN [Production].[ProductSubcategory] AS PSC
ON
P.ProductSubcategoryID=PSC.ProductSubcategoryID
INNER JOIN [Production].[ProductCategory] AS PC
ON PSC.ProductCategoryID=PC.ProductCategoryID
```

4 Найти название продукта, отпускную цену продукта, а также последнюю отпускную цену этого продукта (LStReceiptCost), которую можно узнать из таблицы ProductVendor.

```
SELECT P.Name, P.ListPrice, PV.LStReceiptCost
FROM [Production].[Product] AS P INNER JOIN
[PurchASIng].[ProductVendor] AS PV
ON P.ProductID=PV.ProductID
```

5 Найти название продукта, отпускную цену продукта, а также последнюю отпускную цену этого продукта (LStReceiptCost), которую можно узнать из таблицы ProductVendor, для таких продуктов, у которых отпускная цена оказалась ниже последней отпускной цены у поставщика, исключив те товары, для которых отпускная цена равна нулю.

```
SELECT P.Name, P.ListPrice, PV.LStReceiptCost
FROM [Production].[Product] AS P INNER JOIN
```



```
[PurchASINg].[ProductVendor] AS PV
ON P.ProductID=PV.ProductID
WHERE P.ListPrice!=0 AND P.ListPrice<PV.LASReceiptCost
```

6 Найти количество товаров, которые поставляют поставщики с самым низким кредитным рейтингом (CreditRatINg принимает целые значение от минимального, равного 1, до максимального, равного 5).

```
SELECT COUNT(DISTINCT PV.ProductID)
FROM [PurchASINg].[ProductVendor] AS PV INNER JOIN [PurchAS-
INg].[Vendor] AS V
ON PV.BusINessEntityID=V.BusINessEntityID
WHERE [CreditRatINg]=1
```

7 Найти, сколько товаров приходится на каждый кредитный рейтинг, т.е. сформировать таблицу, первая колонка которой будет содержать номер кредитного рейтинга, вторая – количество товаров, поставляемых всеми поставщиками, имеющими соответствующий кредитный рейтинг. Необходимо сформировать универсальный запрос, который будет валидным и в случае появления новых значений кредитного рейтинга.

```
SELECT [CreditRatINg], COUNT(DISTINCT PV.ProductID)
FROM [PurchASINg].[ProductVendor] AS PV INNER JOIN [PurchAS-
INg].[Vendor] AS V
ON PV.BusINessEntityID=V.BusINessEntityID
GROUP BY [CreditRatINg]
```

8 Найти номера первых трех подкатегорий (ProductSubcategoryID) с наибольшим количеством наименований товаров.

```
SELECT TOP 3 [ProductSubcatego-
ryID] FROM [Production].[Product]
WHERE [ProductSubcategoryID] IS NOT NULL
GROUP BY [ProductSubcategoryID]
ORDER BY COUNT(*) DESC
```

9 Получить названия первых трех подкатегорий с наибольшим количеством наименований товаров.

```
SELECT TOP 3 PC.ProductCategoryID
FROM [Production].[Product] AS P INNER JOIN [Pro-
duction].[ProductSubcategory] AS PSC
ON
P.ProductSubcategoryID=PSC.ProductSubcategoryID
INNER JOIN [Production].[ProductCategory] AS PC
ON PSC.ProductCategoryID=PC.ProductCategoryID
GROUP BY PC.ProductCategoryID
ORDER BY COUNT(*) DESC
SELECT top 3 psc.name, count(*)
FROM [Production].[Product] AS p INNER JOIN [Pro-
duction].[ProductSubcategory] AS psc
ON p.ProductSubcategoryID=psc.ProductSubcategoryID
WHERE p.ProductSubcategoryID is NOT null
GROUP BY p.ProductSubcategoryID, psc.Name
ORDER BY count(*) desc
```

10 Вычислить среднее количество товаров, приходящихся на одну подкатегорию, с точностью минимум до одной десятой.

Вариант 1

```
SELECT 1.0*COUNT(*)/COUNT(DISTINCT [ProductSubcategoryID])
FROM [Production].[Product]
WHERE [ProductSubcategoryID] IS NOT NULL
```

Вариант 2

```
SELECT CAST(COUNT(*) AS
          FLOAT)/COUNT(DISTINCT
[ProductSubcategoryID])
FROM [Production].[Product]
WHERE [ProductSubcategoryID] IS NOT NULL
```

Вариант 3

```
SELECT CAST((CAST(count(p.ProductID) AS
                float)/count(DISTINCT p.ProductSubcategoryID))
AS decimal(6,3))
FROM [Production].[Product] AS p
WHERE p.ProductSubcategoryID is NOT null
```

11 Вычислить среднее количество товаров, приходящихся на одну категорию, в целых числах.

```
SELECT COUNT(*)/COUNT(DISTINCT PC.ProductCategoryID)
FROM [Production].[Product] AS P INNER JOIN [Production].[ProductSubcategory] AS PSC
ON
P.ProductSubcategoryID=PSC.ProductSubcategoryID
RIGHT JOIN [Production].[ProductCategory] AS PC
ON PSC.ProductCategoryID=PC.ProductCategoryID
```

12 Найти количество цветов товаров, приходящихся на каждую категорию, без учета товаров, для которых цвет не определен.

```
SELECT COUNT(DISTINCT [Color])
FROM [Production].[Product] AS P INNER JOIN [Production].[ProductSubcategory] AS PSC
ON
P.ProductSubcategoryID=PSC.ProductSubcategoryID
RIGHT JOIN [Production].[ProductCategory] AS PC
ON PSC.ProductCategoryID=PC.ProductCategoryID
GROUP BY PC.ProductCategoryID
```

13 Найти средний вес продуктов. Просмотреть таблицу продуктов и убедиться, что есть продукты, для которых вес не определен. Модифицировать запрос так, чтобы при нахождении среднего веса продуктов те продукты, для которых вес не определен, считались как продукты с весом 10.

```
SELECT AVG(ISNULL([Weight],10))
FROM [Production].[Product]
```

14 Вывести названия продуктов и период их активных продаж (период между SellStartDate и SellEndDate) в днях, отсортировав по уменьшению времени продаж. Если

продажи идут до сих пор и SellEndDate не определен, то считать периодом продаж число дней с начала продаж и по текущие сутки.

```
SELECT [Name], [SellStartDate], [SellEndDate],
DATEDIFF(D, [SellStartDate], ISNULL([SellEndDate], GETDATE()))
FROM [Production].[Product]
WHERE [SellStartDate] IS NOT NULL
```

15 Разбить продукты по количеству символов в названии, и для каждой группы определить количество продуктов.

```
SELECT LEN([Name]), COUNT(*)
FROM [Production].[Product]
GROUP BY LEN([Name])
```

16 Найти для каждого поставщика количество подкатегорий продуктов, к которым относятся продукты, поставляемые им, без учета ситуации, когда продукт не относится ни к какой подкатегории.

```
SELECT PV.BusinessEntityID, COUNT(DISTINCT P.ProductSubcategoryID)
FROM [Production].[Product] AS P INNER JOIN [PurchAS-
ING].[ProductVendor] AS PV
ON P.ProductID=PV.ProductID
WHERE P.ProductSubcategoryID IS NOT NULL
GROUP BY PV.BusinessEntityID
```

17 Проверить, есть ли продукты с одинаковым названием, если есть, то вывести эти названия.

Вариант 1

```
SELECT P1.Name
FROM [Production].[Product] AS P1, [Production].[Product] AS P2
WHERE P1.ProductID!=P2.ProductID AND
P1.Name=P2.Name
```

Вариант 2

```
SELECT [Name]
FROM [Production].[Product]
GROUP BY [ProductID], [Name]
HAVING COUNT(*)>1
```

18 Найти первые 10 самых дорогих товаров, с учетом ситуации, когда цена цены у некоторых товаров могут совпадать.

```
SELECT TOP 10 WITH TIES [Name]
FROM [Production].[Product]
ORDER BY [ListPrice] DESC
```

19 Найти первые 10 процентов самых дорогих товаров, с учетом ситуации, когда цены у некоторых товаров могут совпадать.

```
SELECT TOP 10 PERCENT WITH TIES [Name]
FROM [Production].[Product]
ORDER BY [ListPrice] DESC
```

20 Найти первых трех поставщиков, отсортированных по количеству поставляемых товаров, с учетом ситуации, что количество поставляемых товаров может совпадать для разных поставщиков.

```
SELECT TOP 3 WITH TIES PV.BusinessEntityID
FROM [Production].[Product] AS P INNER JOIN [Pur-
chasing].[ProductVendor] AS PV
ON P.ProductID=PV.ProductID
GROUP BY PV.BusinessEntityID
ORDER BY COUNT(P.ProductID)
DESC
```

Задания для самостоятельной работы

- 1 Найти и вывести на экран название продуктов и название категорий товаров, к которым относится этот продукт, с учетом того, что в выборку попадут только товары с цветом Red и ценой не менее 100.
- 2 Вывести на экран названия подкатегорий с совпадающими именами.
- 3 Вывести на экран название категорий и количество товаров в данной категории.
- 4 Вывести на экран название подкатегории, а также количество товаров в данной подкатегории с учетом ситуации, что могут существовать подкатегории с одинаковыми именами.
- 5 Вывести на экран название первых трех подкатегорий с небольшим количеством товаров.
- 6 Вывести на экран название подкатегории и максимальную цену продукта с цветом Red в этой подкатегории.
- 7 Вывести на экран название поставщика и количество товаров, которые он поставляет.
- 8 Вывести на экран название товаров, которые поставляются более чем одним поставщиком.
- 9 Вывести на экран название самого продаваемого товара.
- 10 Вывести на экран название категории, товары из которой продаются наиболее активно.
- 11 Вывести на экран названия категорий, количество подкатегорий и количество товаров в них.
- 12 Вывести на экран номер кредитного рейтинга и количество товаров, поставляемых компаниями, имеющими этот кредитный рейтинг.

Лабораторная работа 4

Цель работы: использование подзапросов.

При написании запросов может возникнуть необходимость в данных, полученных на предыдущем этапе. Встает вопрос об актуальности данных. Рассмотрим следующий пример. Необходимо найти название самого дорогого товара с цветом Red. Пользователь, не знакомый с механизмом использования подзапросов, может написать следующий набор запросов.

```
SELECT MAX([ListPrice]) FROM [Production].[Product]
WHERE [Color]='Red'
```

Данный запрос вернет таблицу, одна строка и один столбец без имени, со значением 3578,27. Подставив полученный результат в следующий запрос, пользователь получит искомые данные.

```
SELECT [Name]
FROM [Production].[Product]
WHERE [Color]='Red' AND [ListPrice]=3578.27
```

При выполнении данных операций пользователь, во-первых, вынужден выполнять проме-

жуточные действия вручную, во-вторых, актуальность полученных данных может быть поставлена под сомнение.

Использование подзапросов решит обе эти проблемы.

```
SELECT [Name]
FROM [Production].[Product]
WHERE [Color]='Red'AND [ListPrice]=
(SELECT MAX([ListPrice]) FROM [Production].[Product]
WHERE [Color]='Red')
```

В данном случае показан простейший вариант использования подзапроса. Стоит отметить, что использован оператор сравнения на равенство, так как функция MAX гарантированно возвращает одно значение.

Допускается использование логических операторов для сравнения скалярного значения с результатом выполнения подзапроса. Рассмотрим несколько примеров.

Необходимо получить список товаров, цвет которых может быть любой, кроме Red, а цена равна цене любого товара с цветом Red. Для этого можно использовать запрос следующего вида:

```
SELECT [Name]
FROM [Production].[Product]
WHERE [Color]!='Red'AND [ListPrice] = ANY
(SELECT [ListPrice] FROM [Production].[Product]
WHERE [Color]='Red')
```

Оператор сравнения использован вместе с логическим оператором ANY, так как подзапрос возвращает потенциально более одного значения. Логический оператор ANY сравнивает скалярное значение с набором значений, состоящим из одного столбца, и условие сравнения должно быть выполнено хотя бы для одного значения из набора.

Необходимо получить список товаров, цена которых больше цены любого из товаров с цветом Red.

```
SELECT [Name]
FROM [Production].[Product]
WHERE [ListPrice] >ALL
(SELECT [ListPrice] FROM [Production].[Product]
WHERE [Color]='Red')
```

Логический оператор ALL сравнивает скалярное значение с набором значений, состоящим из одного столбца, и условие сравнения должно быть выполнено для каждого значения из набора.

Необходимо получить название товаров, чей цвет совпадает с цветом одного из товаров, чья цена больше 3000.

```
SELECT [Name]
FROM [Production].[Product]
WHERE [Color] IN
(SELECT [Color] FROM [Production].[Product]
WHERE [ListPrice]>3000)
```

Логический оператор IN определяет, совпадает ли указанное значение с одним из значений, содержащихся во вложенном запросе или списке.

Подзапросы могут сами содержать подзапросы. Следующий пример находит название категории, где содержится самый дорогой товар.

```

SELECT [Name]
FROM [Production].[ProductCategory] WHERE
[ProductCategoryID] IN (SELECT
[ProductCategoryID]
FROM [Production].[ProductSubcategory] WHERE
[ProductSubcategoryID] IN (SELECT
[ProductSubcategoryID]
FROM [Production].[Product] WHERE
[ListPrice] = (SELECT
MAX([ListPrice])
FROM [Production].[Product])))

```

Данную задачу можно было бы решить с помощью соединения таблиц, упорядочивания и выражения TOP, но использование подзапросов в данном случае предпочтительней, так как это позволяет избежать ресурсоемкой операции соединения. Это утверждение справедливо только для ситуаций, когда подзапрос простой.

Запрос может использовать более одного подзапроса на одном уровне вложенности. Например, необходимо получить с помощью одного запроса список товаров, у которых цвет совпадает с цветом самого дорогого товара, и стиль совпадает со стилем самого дорогого товара.

```

SELECT [Name]
FROM [Production].[Product]
WHERE [Color] IN
(SELECT [Color]
FROM [Production].[Product] WHERE
[ListPrice] = (SELECT
MAX([ListPrice])
FROM [Production].[Product]))
AND
[Style] IN (SELECT [Style]
FROM [Production].[Product] WHERE
[ListPrice] = (SELECT
MAX([ListPrice])
FROM [Production].[Product]))

```

Все рассмотренные ранее подзапросы являлись простыми, они возвращали фиксированный, неизменный набор данных, вне зависимости от того, с какой из строк запроса верхнего уровня в данный момент работает СУБД.

Подзапросы также используются для формирования выборки с использованием конструкции GROUP BY HAVING. Допустим, необходимо найти номер подкатегории товаров с наибольшим количеством товаров. Данный запрос можно реализовать несколькими способами, в том числе с использованием подзапроса.

```

SELECT [ProductSubcategoryID]
FROM [Production].[Product]
GROUP BY [ProductSubcategoryID]
HAVING COUNT(*) =
  (SELECT TOP 1 COUNT(*)
   FROM [Production].[Product]
   GROUP BY [ProductSubcategoryID]
   ORDER BY 1 DESC)

```

Рассмотрим следующую задачу. Необходимо получить список самых дорогих товаров в каждой из подкатегорий. Подобную задачу можно решить с использованием сложного, коррелирующего или связанного, подзапроса. Коррелирующим подзапросом называют такой подзапрос, который формирует связанную выборку, зависимую от данных внешнего запроса. Фактически коррелирующий подзапрос выполняется для каждой строки запроса верхнего уровня.

```

SELECT [Name]
FROM [Production].[Product] AS P1
WHERE [ListPrice] =
  (SELECT MAX([ListPrice])
   FROM [Production].[Product] AS P2
   WHERE P1.ProductSubcategoryID=P2.ProductSubcategoryID)

```

Связанные подзапросы могут использоваться для формирования выводимого столбца. Например, следующий запрос возвращает название продукта и название подкатегории, к которой он относится.

```

SELECT [Name],
  (SELECT [Name]
   FROM [Production].[ProductSubcategory] AS PS
   WHERE P1.ProductSubcategoryID=PS.ProductSubcategoryID)
FROM [Production].[Product] AS P1

```

Примеры запросов с решениями

- 1 Найти название подкатегории с наибольшим количеством продуктов, без учета продуктов, для которых подкатегория не определена (еще одна возможная реализация).

```

SELECT [Name]
FROM [Production].[ProductSubcategory]
WHERE [ProductSubcategoryID] IN
  (SELECT [ProductSubcategoryID]
   FROM [Production].[Product]
   WHERE [ProductSubcategoryID] IS NOT NULL
   GROUP BY [ProductSubcategoryID]
   HAVING COUNT(*) = (SE-
    LECT TOP 1 COUNT(*)
   FROM [Production].[Product]
   WHERE [ProductSubcategoryID] IS NOT NULL
   GROUP BY [ProductSubcategoryID]
   ORDER BY 1 DESC
  )
)

```

2 Вывести на экран такого покупателя, который каждый раз покупал только одну номенклатуру товаров, не обязательно в одинаковых количествах, т.е. у него всегда был один и тот же «список покупок».

Вариант 1

```
SELECT [CustomerID], count(*)
FROM [Sales].[SalesORDERHeader] AS
SOH1 GROUP BY [CustomerID]
HAVING count(*)>1 AND count(*)=all
(SELECT count(*)
FROM [Sales].[SalesORDERHeader] AS SOH INNER JOIN
[Sales].[SalesORDERDetail] AS SOD
ON soh.SalesORDERID=sod.SalesORDERID
GROUP BY soh.[CustomerID], sod.ProductID
HAVING soh.CustomerID=soh1.CustomerID)
```

Вариант 2

```
SELECT soh.CustomerID, p.Name
FROM [Sales].[SalesORDERHeader] AS SOH INNER JOIN
[Sales].[SalesORDERDetail] AS sod
ON soh.SalesORDERID=sod.SalesORDERID INNER JOIN
[Production].[Product] AS p
ON sod.ProductID=p.ProductID
WHERE soh.CustomerID IN ( SELECT [CustomerID]
FROM [Sales].[SalesORDERHeader] AS
SOH1 GROUP BY [CustomerID]
HAVING count(*)>1 AND count(*)=all
(SELECT count(*)
FROM [Sales].[SalesORDERHeader] AS SOH INNER JOIN
[Sales].[SalesORDERDetail] AS SOD
ON soh.SalesORDERID=sod.SalesORDERID
GROUP BY soh.[CustomerID], sod.ProductID
HAVING soh.CustomerID=soh1.CustomerID))
ORDER BY 1
```

Вариант 3

```
SELECT soh1.CustomerID
FROM [Sales].[SalesORDERDetail] AS sod1 INNER JOIN
[Sales].[SalesORDERHeader] AS soh1
ON sod1.SalesORDERID=soh1.SalesORDERID
GROUP BY soh1.CustomerID
HAVING count(soh1.SalesORDERID)>1 AND
count(DISTINCT sod1.ProductID)>1
AND count(DISTINCT sod1.ProductID)=all
(
SELECT count(*)
FROM [Sales].[SalesORDERDetail] AS sod2 INNER JOIN
[Sales].[SalesORDERHeader] AS soh2
```



```

ON sod2.SalesORDERID=soh2.SalesORDERID
GROUP BY soh2.CustomerID,
sod2.SalesORDERID HAVING
soh2.CustomerID=soh1.CustomerID
)

```

3 Вывести на экран следующую информацию: название товара (первая колонка), количество покупателей, покупавших этот товар (вторая колонка), количество покупателей, совершавших покупки, но не покупавших товар из первой колонки (третья колонка).

```

SELECT
p.[ProductID],
(SELECT count(DISTINCT soh.CustomerID)
FROM [Sales].[SalesORDERDetail] AS sod INNER JOIN
[Sales].[SalesORDERHeader] AS soh
ON sod.SalesORDERID=soh.SalesORDERID
WHERE sod.ProductID=p.ProductID), (SE-
LECT count(DISTINCT soh.CustomerID)
FROM [Sales].[SalesORDERDetail] AS sod INNER JOIN
[Sales].[SalesORDERHeader] AS soh
ON sod.SalesORDERID=soh.SalesORDERID
WHERE soh.CustomerID NOT IN
(SELECT DISTINCT soh.CustomerID
FROM [Sales].[SalesORDERDetail] AS sod INNER JOIN
[Sales].[SalesORDERHeader] AS soh
ON sod.SalesORDERID=soh.SalesORDERID WHERE
sod.ProductID=p.ProductID))
FROM [Production].[Product] AS p

```

4 Найти такие товары, которые были куплены более чем одним покупателем, при этом все покупатели этих товаров покупали товары только из одной подкатегории.

```

SELECT name
FROM [Production].[Product]
WHERE ProductID IN(
SELECT sod.ProductID
FROM [Sales].[SalesORDERDetail] AS sod JOIN
[Sales].[SalesORDERHeader] AS soh
ON sod.SalesORDERID=soh.SalesORDERID
WHERE soh.CustomerID IN(
SELECT soh.CustomerID
FROM [Sales].[SalesORDERDetail] AS sod JOIN
[Sales].[SalesORDERHeader] AS soh
ON sod.SalesORDERID=soh.SalesORDERID JOIN [Pro-
duction].[Product] AS p ON
sod.ProductID=p.ProductID
GROUP BY soh.CustomerID
HAVING count(DISTINCT p.ProductSubcategoryID)=1)
GROUP BY sod.ProductID
HAVING count(DISTINCT soh.CustomerID)>1)

```

5 Найти покупателя, который каждый раз имел разный список товаров в чеке (по номенклатуре).

```
SELECT DISTINCT CustomerID
FROM
[Sales].[SalesORDERHeader]
WHERE CustomerID NOT IN ( SE-
LECT soh.Customerid
FROM [Sales].[SalesORDERDetail] AS sod JOIN
[Sales].[SalesORDERHeader] AS soh
ON sod.SalesORDERID=soh.SalesORDERID
WHERE
exists(SELECT ProductID
FROM [Sales].[SalesORDERDetail] AS sod1 JOIN
[Sales].[SalesORDERHeader] AS soh1
ON sod.SalesORDERID=soh.SalesORDERID
WHERE soh1.CustomerID=soh.CustomerID AND
sod1.ProductID=sod.ProductID
```

AN

```
D
sod.SalesORDERID!=sod1.SalesORDERID
))
```

6 Найти такого покупателя, что все купленные им товары были куплены только им и никогда не покупались другими покупателями.

Вариант 1

```
SELECT soh1.CustomerID
FROM[Sales].[SalesORDERDetail] AS sod1 INNER
JOIN [Sales].[SalesORDERHeader] AS soh1
ON sod1.SalesORDERID=soh1.SalesORDERID
GROUP BY soh1.CustomerID
HAVING count(DISTINCT sod1.productid)= (SE-
LECT count(DISTINCT sod.ProductID)
FROM [Sales].[SalesORDERDetail] AS sod INNER JOIN
[Sales].[SalesORDERHeader] AS soh
ON
sod.SalesORDERID=soh.SalesORDERID
WHERE
soh.CustomerID=soh1.CustomerID AND
sod.ProductID IN
(SELECT sod.ProductID
FROM[Sales].[SalesORDERDetail] AS sod INNER JOIN
[Sales].[SalesORDERHeader] AS soh
ON sod.SalesORDERID=soh.SalesORDERID
GROUP BY sod.ProductID
HAVING count(DISTINCT soh.CustomerID)=1))
```

Вариант 2

```
SELECT DISTINCT soh.CustomerID
```

```

FROM [Sales].[SalesORDERHeader] AS soh
WHERE soh.CustomerID NOT IN(
SELECT DISTINCT soh.CustomerID
FROM [Sales].[SalesORDERDetail] AS sod JOIN
[Sales].[SalesORDERHeader] AS soh
ON sod.SalesORDERID=soh.SalesORDERID
WHERE ProductID NOT IN(
SELECT sod.ProductID
FROM [Sales].[SalesORDERDetail] AS sod JOIN
[Sales].[SalesORDERHeader] AS soh
ON sod.SalesORDERID=soh.SalesORDERID
GROUP BY sod.ProductID
HAVING count(DISTINCT soh.CustomerID)=1))

```

Задания для самостоятельной работы

- 1 Найти название самого продаваемого продукта.
- 2 Найти покупателя, совершившего покупку на самую большую сумму, считая сумму покупки исходя из цены товара без скидки (UnitPrice).
- 3 Найти такие продукты, которые покупал только один покупатель.
- 4 Вывести список продуктов, цена которых выше средней цены товаров в подкатегории, к которой относится товар.
- 5 Найти такие товары, которые были куплены более чем одним покупателем, при этом все покупатели этих товаров покупали товары только одного цвета и товары не входят в список покупок покупателей, купивших товары только двух цветов.
- 6 Найти такие товары, которые были куплены такими покупателями, у которых они присутствовали в каждой их покупке.
- 7 Найти покупателей, у которых есть товар, присутствующий в каждой покупке/чеке.
- 8 Найти такой товар или товары, которые были куплены не более чем тремя различными покупателями.
- 9 Найти все товары, такие что их покупали всегда с товаром, цена которого максимальна в своей категории.
- 10 Найти номера тех покупателей, у которых есть как минимум два чека, и каждый из этих чеков содержит как минимум три товара, каждый из которых как минимум был куплен другими покупателями три раза.
- 11 Найти все чеки, в которых каждый товар был куплен дважды этим же покупателем.
- 12 Найти товары, которые были куплены минимум три раза различными покупателями.
- 13 Найти такую подкатегорию или подкатегории товаров, которые содержат более трех товаров, купленных более трех раз.
- 14 Найти те товары, которые не были куплены более трех раз, и как минимум дважды одним и тем же покупателем.

Лабораторная работа 5

Цель работы: использование результатов выборки как источник данных, использование обобщенного табличного выражения (ОТВ).

В подавляющем большинстве случаев операция выборки предполагает, что источником данных будут служить таблицы. В некоторых случаях это создает определенные неудобства, в частности, не позволяет получить выборку из результата выполнения предыдущего

запроса. Решить эту задачу можно несколькими способами.

Представление. Представление является именованным результатом выполнения запроса. Поскольку представление – это объект базы данных, то для его создания и использования требуются дополнительные права.

Альтернативным вариантом, не требующим дополнительных прав, является использование запросов как источников данных или обобщенных табличных выражений.

Использование запроса как источника данных подразумевает, что пользователь создает запрос, дает ему псевдоним и использует этот псевдоним аналогично имени таблицы.

Рассмотрим сложный и объемный пример. Необходимо найти для каждой подкатегории количество товаров, у которых цена выше средней цены в подкатегории, и количество товаров, у которых цена ниже средней цены в подкатегории. Оформить вывод в виде трех столбцов: номер подкатегории, первый показатель, второй показатель. Решить подобную задачу традиционным способом достаточно сложно, но она относительно легко решается с использованием псевдонимов для результата выполнения запроса.

```
SELECT T1.PS, T1.c, T2.c FROM
(SELECT COUNT(*) AS c, [ProductSubcategoryID] AS PS
FROM [Production].[Product] AS P
WHERE [ListPrice]<(SELECT avg([ListPrice])
FROM [Production].[Product] AS PT
WHERE pt.ProductSubcategoryID=p.ProductSubcategoryID)
GROUP BY [ProductSubcategoryID]) AS T1 INNER JOIN
(SELECT count(*) AS c, [ProductSubcategoryID] AS PS
FROM [Production].[Product] AS P
WHERE [ListPrice]>=(SELECT avg([ListPrice])
FROM [Production].[Product] AS PT
WHERE pt.ProductSubcategoryID=p.ProductSubcategoryID)
GROUP BY [ProductSubcategoryID]) AS T2 ON
T1.PS=T2.PS
```

В данном примере созданы два независимых запроса, один из которых находит номер категории и количество товаров, цена у которых ниже средней цены в той же категории, к которой он относится, второй – такой же, но знак заменен на «больше или равно». Каждый запрос получил свой псевдоним, T1 и T2 соответственно, и каждому столбцу в запросе дан свой псевдоним. Запрос верхнего уровня использует запросы T1 и T2 точно так же, как обычные таблицы, проводит операцию INNER JOIN и выводит искомый результат.

К сожалению, использование такого подхода удобно, когда запрос, формирующий источник данных, не очень длинный. В противном случае читаемость кода резко падает, и в этом случае удобнее использовать обобщенное табличное выражение. Также обобщенное табличное выражение можно использовать при выполнении однократной операции модификации данных.

Общий синтаксис:

```
WITH <имя ОТВ>
( столбец 1, столбец 2, ... ]
AS
(запрос)
```

В рекурсивных ОТВ недопустимо использовать операцию упорядочивания, за исключени-

ем случаев использования инструкции TOP.

Рассмотрим пример, в котором необходимо найти количество чеков, приходящихся на одного покупателя на каждый год.

```
WITH Sales_CTE (SalesPersonID, SalesORDERID, SalesYear)
AS
(
SELECT SalesPersonID, SalesORDERID, YEAR(ORDERDate) AS SalesYear
FROM Sales.SalesORDERHeader
WHERE SalesPersonID IS NOT NULL
)
SELECT SalesPersonID, COUNT(SalesORDERID) AS TotalSales, SalesYear
FROM Sales_CTE
GROUP BY SalesYear, SalesPersonID
ORDER BY SalesPersonID, SalesYear
```

В данном пример определено ОТВ с именем Sales_CTE, которое использовано как источник данных для запроса.

Если необходимо использовать несколько ОТВ в одном запросе, то их можно определить следующим образом.

```
WITH <имя ОТВ1>
( столбец 1, столбец 2, ... ]
AS
(запрос) ,
<имя ОТВ2>
( столбец 1, столбец 2, ... ]
AS
(запрос)
```

Примеры запросов с решениями

1 Найти покупателя, который каждый раз имел разный список товаров в чеке (по номенклатуре)

```
SELECT tmp.c
FROM
(SELECT soh.CustomerID AS c
, soh.SalesORDERID AS o
, CHECKSUM_AGG(sod.ProductID) AS ch
FROM [Sales].[SalesORDERDetail] AS sod JOIN
[Sales].[SalesORDERHeader] AS soh
ON sod.SalesORDERID=soh.SalesORDERID
GROUP BY soh.CustomerID, soh.SalesORDERID) tmp
GROUP BY tmp.c
HAVING count(tmp.ch)=count(DISTINCT tmp.ch)
AND count(tmp.ch)>1
```

2 Найти пары таких покупателей, что список названий товаров, которые они когда-либо покупали, не пересекается ни в одной позиции.

```
SELECT top 3 t1.c, t2.c
FROM
(SELECT soh.CustomerID AS c,
```

```

sod.ProductID AS p
FROM [Sales].[SalesORDERDetail] AS sod JOIN
[Sales].[SalesORDERHeader] AS soh
ON sod.SalesORDERID=sod.SalesORDERID) t1,
(SELECT soh.CustomerID AS c,
sod.ProductID AS p
FROM [Sales].[SalesORDERDetail] AS sod JOIN
[Sales].[SalesORDERHeader] AS soh
ON sod.SalesORDERID=sod.SalesORDERID) t2
WHERE t1.p!=all(SELECT sod.ProductID AS p
FROM [Sales].[SalesORDERDetail] AS sod JOIN
[Sales].[SalesORDERHeader] AS soh
ON sod.SalesORDERID=sod.SalesORDERID
WHERE soh.CustomerID=t2.c)

```

3 Вывести номера продуктов, таких, что их цена выше средней цены продукта в подкатегории, к которой относится продукт. Запрос реализовать двумя способами. В одном из решений допускается использование обобщенного табличного выражения.

Вариант 1

```

SELECT p1.ProductID
FROM [Production].[Product] AS p1
WHERE p1.ListPrice>
(
SELECT avg(p2.[ListPrice])
FROM [Production].[Product] AS p2
WHERE p2.ProductSubcategoryID=p1.ProductSubcategoryID
)

```

Вариант 2

```

with tmp (pscid, acgLP) AS
(SELECT p.ProductSubcategoryID, avg([ListPrice])
FROM [Production].[Product] AS p
GROUP BY p.ProductSubcategoryID)

```

```

SELECT p.ProductID
FROM [Production].[Product] AS p JOIN
tmp ON
p.ProductSubcategoryID=tmp.pscid WHERE
[ListPrice]>tmp.acgLP

```

Задания для самостоятельной работы

- 1 Найти среднее количество покупок на чек для каждого покупателя (2 способа).
- 2 Найти для каждого продукта и каждого покупателя соотношение количества фактов покупки данного товара данным покупателем к общему количеству фактов покупки товаров данным покупателем
- 3 Вывести на экран следующую информацию: Название продукта, Общее количество фактов покупки этого продукта, Общее количество покупателей этого продукта

- 4 Вывести для каждого покупателя информацию о максимальной и минимальной стоимости одной покупки, чеке, в виде таблицы: номер покупателя, максимальная сумма, минимальная сумма.
- 5 Найти номера покупателей, у которых не было нет ни одной пары чеков с одинаковым количеством наименований товаров.
- 6 Найти номера покупателей, у которых все купленные ими товары были куплены как минимум дважды, т.е. на два разных чека.

Лабораторная работа 6

Цель работы: Использование оконных функций и предложения OVER.

Предложение OVER определяет секционирование и упорядочение набора строк до применения соответствующей оконной функции.

```
OVER (
[ <PARTITION BY столбец> ]
[ <ORDER BY столбец> ]
[ <ROW или RANGE столбец> ]
)
```

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 логический порядок строк в каждой секции результирующего набора.

ROW or RANGE ограничивает строки в пределах секции, указывая начальную и конечную точки.

Параметр CURRENT ROW указывает, что окно начинается или заканчивается на текущей строке при использовании совместно с предложением ROWS или что окно заканчивается на текущем значении при использовании с предложением RANGE. CURRENT ROW может быть задана и как начальная, и как конечная точка.

Параметр BETWEEN <граница рамки окна > AND <граница рамки окна > используется совместно с предложением ROWS или RANGE для указания нижней (начальной) или верхней (конечной) граничной точки окна. <граница рамки окна> определяет граничную начальную точку, а <граница рамки окна> определяет граничную конечную точку.

Параметр UNBOUNDED FOLLOWING указывает, что окно заканчивается на последней строке секции. UNBOUNDED FOLLOWING может быть указано только как конечная точка окна.

Пример:

```
RANGE BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING
```

В данном случае параметр определяет, что окно начинается на текущей строке и заканчивается на последней строке секции.

```
ROWS BETWEEN 2 FOLLOWING AND 10 FOLLOWING
```

В этом примере параметр определяет, что окно начинается на второй строке после текущей и заканчивается на десятой строке после текущей строки. Эта спецификация не допускается в предложении RANGE.

Фактически предложение OVER виртуально разбивает выбранные строки на наборы, окна, определяемые в условии PARTITION BY, упорядочивает эти строки по столбцам определенным ORDER BY. В рамках этих наборов, окон, выполняются те или иные агрегирующие

щие, статистические и иные функции. Результат выполнения этих функций формирует отдельный столбец с одинаковыми значениями для каждой строки в наборе, окне. Однако можно для каждой строки в наборе формировать свое значение функции, для чего используют параметр ROW или RANGE, который определяет диапазон строк, в наборе, окне, с которыми будет работать функция.

```
SELECT      SalesORDERID,      ProductID,      ORDERQty
            ,SUM(ORDERQty) OVER(PARTITION BY SalesORDERID) AS Total
,AVG(ORDERQty) OVER(PARTITION BY SalesORDERID) AS "Avg"
,COUNT(ORDERQty) OVER(PARTITION BY SalesORDERID) AS "Count"
,MIN(ORDERQty) OVER(PARTITION BY SalesORDERID) AS "MIN"
,MAX(ORDERQty) OVER(PARTITION BY SalesORDERID) AS "Max"
FROM Sales.SalesORDERDetail
WHERE SalesORDERID IN(43659,43664)
```

Следующий запрос использует приведение типа.

```
SELECT SalesORDERID, ProductID, ORDERQty
, SUM(ORDERQty) OVER(PARTITION BY SalesORDERID) AS Total
,CAST(1. * ORDERQty / SUM(ORDERQty) OVER(PARTITION BY SalesORDERID)
*100 AS DECIMAL(5,2))AS "Percent BY ProductID"
FROM Sales.SalesORDERDetail
WHERE SalesORDERID IN(43659,43664)
```

Можно выполнять функцию не на всем наборе, но на части набора. Часть определяется в зависимости от строки и ее положения в наборе.

```
SELECT BusinessEntityID, TerritoryID
, CONVERT(varchar(20),SalesYTD,1) AS SalesYTD
, DATEPART(yy,ModifiedDate) AS SalesYear
, CONVERT(varchar(20),SUM(SalesYTD) OVER (PARTITION BY TerritoryID
ORDER BY DATEPART(yy,ModifiedDate)
ROWS BETWEEN CURRENT ROW AND 1 FOLLOWING ),1) AS CumulativeTotal
FROM Sales.SalesPerson
WHERE TerritoryID IS NULL OR TerritoryID < 5
```

Аналитические функции.

Функция FIRST_VALUE возвращает первое значение из упорядоченного набора значений.

Синтаксис

```
FIRST_VALUE ( [scalar_expression ] ) OVER ( [ partition_BY_clause ]
ORDER_BY_clause [ rows_range_clause ] )
```

scalar_expressiON может быть столбцом, вложенным запросом.

Получение имени самого дешевого продукта в заданной категории продуктов.

```
SELECT Name, ListPrice,
FIRST_VALUE(Name) OVER (ORDER BY ListPrice ASC) AS LeAStExpensive
FROM Production.Product
WHERE ProductSubcategoryID = 37
```

Функция LAST_VALUE возвращает последнее значение из упорядоченного набора значений.

Синтаксис

```
LAST_VALUE ( [ scalar_expression ] ) OVER ( [ partition_BY_clause ]
ORDER_BY_clause rows_range_clause )
```


scalar_expressiON может быть столбцом, вложенным запросом.

Возвращение даты найма последнего сотрудника каждого отдела для указанной заработной платы (Rate). Предложение PARTITION BY разделяет сотрудников по отделам, а функция LAST_VALUE применяется к каждой секции в отдельности. Предложение ORDER BY, указанное в предложении OVER, определяет логический порядок, в котором функция LAST_VALUE применяется к строкам каждой секции.

```
SELECT Department, LastName, Rate, HireDate,
LAST_VALUE(HireDate) OVER (PARTITION BY Department ORDER BY Rate) AS
LAsTValue
```

```
FROM HumanResources.vEmployeeDepartmentHistory AS
edh INNER JOIN HumanResources.EmployeePayHistory AS
eph ON eph.BusinessEntityID = edh.BusinessEntityID
INNER JOIN HumanResources.Employee AS e
ON e.BusinessEntityID = edh.BusinessEntityID
WHERE Department IN (N'INformation Services',N'Document CONTrol')
```

Функция LAG обеспечивает доступ к строке с заданным физическим смещением перед началом текущей строки.

```
LAG (scalar_expression [,offset] [,default])
OVER ( [ partition_BY_clause ] ORDER_BY_clause )
```

scalar_expressiON – возвращаемое значение основано на указанном смещении.

offset – количество строк до строки перед текущей строкой, из которой необходимо получить значение.

default – возвращаемое значение, когда offset находится за пределами секции. Нахождение квоты для работника за год и предыдущий год.

Квоты менялись несколько раз в год, но для первой установленной в году квоты нет предыдущего значения от 2010 года, их не включают в выборку.

```
SELECT BusinessEntityID, YEAR(QuotaDate) AS SalesYear, SalesQuota AS
CurrentQuota,
LAG(SalesQuota, 1,0) OVER (ORDER BY YEAR(QuotaDate))
AS PreviousQuota
```

```
FROM Sales.SalesPersonQuotaHistory
WHERE BusinessEntityID = 275 AND YEAR(QuotaDate) IN ('2011','2012')
```

Функция LEAD – доступ к строке на заданном физическом смещении после текущей строки.

Синтаксис

```
LEAD ( scalar_expression [ ,offset ] , [ default ] ) OVER ( [
partition_BY_clause ] ORDER_BY_clause )
```

Пример, получение квот продаж для указанного работника за последующие годы.

```
SELECT BusinessEntityID, YEAR(QuotaDate) AS SalesYear, SalesQuota AS
CurrentQuota,
LEAD(SalesQuota, 1,0) OVER (ORDER BY YEAR(QuotaDate)) AS NextQuota
FROM Sales.SalesPersonQuotaHistory
WHERE BusinessEntityID = 275 AND YEAR(QuotaDate) IN ('2011','2012')
```

Функция NTILE распределяет строки упорядоченной секции в заданное количество групп. Группы нумеруются, начиная с единицы. Для каждой строки функция NTILE возвращает номер группы, которой принадлежит строка.

Синтаксис

NTILE (INteger_expressiON) OVER ([<partitiON_BY_clause>] <ORDER_BY_clause>)

INteger_expressiON – положительное целое выражение, указывающее число групп, на которые необходимо разделить каждую секцию.

Функция ROW_NUMBER нумерует выходные данные результирующего набора.

ROW_NUMBER ()

OVER ([PARTITION BY value_expression , ... [n]] ORDER_BY_clause)

Пример использования функций.

```
SELECT p.FirstName, p.LastName
,ROW_NUMBER() OVER (ORDER BY a.PostalCode) AS "Row Number"
,NTILE(4) OVER (ORDER BY a.PostalCode) AS Quartile
,s.SalesYTD
,a.PostalCode
FROM Sales.SalesPerson AS s
INNER JOIN Person.Person AS p
ON s.BusinessEntityID = p.BusinessEntityID
INNER JOIN Person.Address AS a
ON a.AddressID = p.BusinessEntityID
WHERE TerritoryID IS NOT NULL AND SalesYTD <> 0
```

Примеры запросов с решениями

1 Найти долю затрат каждого покупателя на каждый купленный им продукт среди общих его затрат в данной сети магазинов. Можно использовать обобщенное табличное выражение.

```
SELECT [SalesORDERID], p.[ProductID], [ProductSubcategoryID], [ORDERQty]*[UnitPrice],
[ORDERQty]*[UnitPrice]/sum([ORDERQty]*[UnitPrice])
OVER(partition BY [SalesORDERID]
, [ProductSubcategoryID])
FROM [Sales].[SalesORDERDetail] AS SOD INNER JOIN
[Production].[Product] AS p
ON SOD.ProductID=p.ProductID
```

2 Для одного выбранного покупателя вывести, для каждой покупки (чека), разницу в деньгах между этой и следующей покупкой.

Вариант 1

```
with tmp (customer, ORDERid, total) AS (SELECT soh.CustomerID, soh.SalesORDERID,
sum(sod.[ORDERQty]*[UnitPrice]) AS total
FROM [Sales].[SalesORDERHeader] AS SOH INNER JOIN
[Sales].[SalesORDERDetail] AS SOD
ON soh.SalesORDERID=sod.SalesORDERID
GROUP BY soh.CustomerID, soh.SalesORDERID)
SELECT customer, ORDERid, total,
total-LEAD(total,1,0)
OVER(partition BY customer ORDER BY ORDERid)
FROM tmp
```

3 Вывести следующую информацию: номер покупателя, номер чека этого покупателя, отсортированные по покупателям, номерам чека (по возрастанию). Третья колонка должна содержать в каждой своей строке сумму текущего чека покупателя со всеми предыдущими чеками этого покупателя.

Вариант 1

```
with tmp (cus, ord, ORDERsum)
AS (SELECT oh.CustomerID, od.SalesORDERID,
sum(od.[UnitPrice]*[ORDERQty])
FROM [Sales].[SalesORDERDetail] AS OD
INNER JOIN
[Sales].[SalesORDERHeader] AS OH
ON
od.SalesORDERID=oh.SalesORDERID
GROUP BY oh.CustomerID, od.SalesORDERID) SE-
LECT
cus, ord, ORDERsum,
sum(ORDERsum)
OVER(partition BY cus ORDER BY ord desc
RANGE BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING)
FROM tmp
```

Задания для самостоятельной работы

- 1 Найти долю продаж каждого продукта (цена продукта * количество продукта), на каждый чек, в денежном выражении.
- 2 Вывести на экран список продуктов, их стоимость, а также разницу между стоимостью этого продукта и стоимостью самого дешевого продукта в той же подкатегории, к которой относится продукт.
- 3 Вывести три колонки: номер покупателя, номер чека покупателя (отсортированный по возрастанию даты чека) и искусственно введенный порядковый номер текущего чека, начиная с 1, для каждого покупателя.
- 4 Вывести номера продуктов, таких что их цена выше средней цены продукта в подкатегории, к которой относится продукт. Запрос реализовать двумя способами. В одном из решений допускается использование обобщенного табличного выражения.
- 5 Вывести на экран номер продукта, название продукта, а также информацию о среднем количестве этого продукта, приходящихся на три последних по дате чека, в которых был этот продукт.

Перечень вопросов и заданий, выносимых на экзамен

Экзамен проводится в устной форме.

Перечень вопросов:

1. Развитие баз данных.
2. Архитектура современной СУБД.
3. Физическая модель базы данных.
4. Модель сущность-связь.
5. Преобразование модели сущность-связь в физическую модель.
6. Реляционная модель данных. Ключи.
7. Функциональные зависимости: замыкание, эквивалентность и правила вывода.

8. Функциональные зависимости: замыкание атрибутов, неприводимые множества функциональных зависимостей, их построение.
9. Цели и средства нормализации
10. Нормальные формы: первая и вторая
11. Нормальные формы: третья и Бойса-Кодда
12. Многозначные зависимости и четвертая нормальная форма
13. Зависимости соединения и пятая нормальная форма
14. Реляционная алгебра: предназначение и свойства
15. Реляционная алгебра: унарные и множественные операции
16. Реляционная алгебра: соединения
17. Реляционная алгебра: деление и операции над данными
18. Исчисление кортежей и его реляционная полнота
19. Исчисление доменов и его реляционная полнота
20. Datalog и рекурсия
21. Целостность данных. Триггеры
22. Представления и их обновление
23. Управление доступом к данным
24. Подсистема хранения данных.
25. Индексация данных. Упорядоченные и хеш-индексы.
26. Индексация данных. Другие типы индексов. Применение индексов
27. Хранимые процедуры и функции. Сходства и различия.
28. Императивное подмножество SQL
29. Data Control Language
30. Транзакции. Восстановление. Классический алгоритм
31. Транзакции. Восстановление. Алгоритм ARIES
32. Транзакции. Параллельное исполнение. Блокировки.
33. Транзакции. Параллельное исполнение. Уровни изоляции.
34. Этапы обработки запроса. Перезапись запросов.
35. Оптимизация запросов. Выбор структуры исполнения запроса
36. Оптимизация запросов. Выбор методов исполнения запроса
37. Оптимизация запросов. Оценка размера и распределения
38. Секционирование
39. Репликация
40. Распределенные транзакции
41. Распределенные базы данных. Цели и проблемы
42. Иерархические данные. Модель близости
43. *Иерархические данные. Вложенные множества*
44. Иерархические данные. Модель путей
45. Временные данные. Полутемпоральные базы данных
46. Временные данные. Модель интервалов
47. Временные данные. Модель событий
48. ORM. Ключи и ссылки
49. ORM. Наследование
50. ORM. Модель сущность-атрибут-значение

Порядок формирования экзаменационного билета:

Билеты состоят из 2-х вопросов:

- 1 вопрос – с 1 по 25 вопрос из перечня вопросов к экзамену;
- 2 вопрос – с 26 по 50 вопрос из перечня вопросов к экзамену.

Пример экзаменационного билета № 1

1. Вопрос «Модель сущность-связь»
2. Вопрос «ORM. Модель сущность-атрибут-значение»

Таблица 9 - Примеры оценочных средств с ключами правильных ответов

№ п/п	Тип задания	Формулировка задания	Правильный ответ	Время выполнения (в минутах)
ОПК-4. Способен понимать принципы работы современных информационных технологий и использовать их для решения задач профессиональной деятельности.				
1.	Задание закрытого типа	<i>Выберите верные ответы.</i> Какие операции реляционной алгебры являются основными? 1. декартово произведение 2. объединение 3. пересечение 4. деление	1, 2	1-3
2.		<i>Выберите верный ответ.</i> Что такое JOIN? 1. операция суммирования 2. операция создания 3. операция объединения 4. операция группировки	3	1-3
3.		<i>Вставьте пропущенное слово.</i> ... - унарная операция, результатом которой является подмножество кортежей исходного отношения, соответствующих условиям, которые накладываются на значения определённых атрибутов. 1. проекция 2. селекция 3. декартово произведение 4. объединение	2	1-3
4.		<i>Выберите верный ответ.</i> Одна из наиболее часто используемых операций SQL – это 1. проекция 2. селекция 3. декартово произведение 4. объединение	2	1-3
5.		<i>Вставьте пропущенное слово.</i> ... - унарная операция (выполняемая над одним отношением), служащая для выбора подмножества атрибутов из отношения R. 1. проекция 2. селекция 3. декартово произведение 4. объединение	1	1-3
6.	Задание открытого типа	Перечислите операции реляционной алгебры, которые являются основными.	Проекция, селекция, декартово произведение, объединение, разность	3-5
7.		Перечислите операции реляционной алгебры, которые являются дополнительными (зависимыми).	Дополнительными (зависимыми) операциями реляционной алгебры	3-5

№ п/п	Тип задания	Формулировка задания	Правильный ответ	Время выполнения (в минутах)
			являются следующие операции: соединение, пересечение, деление	
8.		Напишите определение операции реляционной алгебры «селекция».	Селекция (selectio n) – это унарная операция, результатом которой является подмножество записей исходной таблицы, удовлетворяющих условиям, которые накладываются на значения определённых полей.	3-5
9.		Какие операции определяет реляционная алгебра?	Операциями реляционной алгебры являются пересечение, разность, произведение, создание проекций, соединение, деление.	3-5
10.		Операция реляционной алгебры пересечение отношений – это ...	Операция реляционной алгебры, в результате которой из отношений P1 и P2 строится новое отношение P3, включающее кортежи, принадлежащие P1 или P2.	3-5
11.	Задание комбинированного типа	<i>Верно ли утверждение</i> Выполнение запроса DROP DATABASE Students приведет к удалению таблицы Students. Ответ обоснуйте.	Утверждение неверно, поскольку выполнение этой команды приведет к <i>полному удалению базы данных Students.</i>	1-3
ПК-6. Способен выполнять работы по созданию и сопровождению информационных систем.				
12.	Задание закрытого типа	<i>Выберите верный ответ.</i> Какая команда используется для создания новой виртуальной таблицы, которая базируется на результатах сделанного ранее SQL запроса? 1. CREATE VIRTUAL TABLE 2. CREATE VIEW 3. ALTER VIEW 4. CREATE TABLE	2	1-3
13.		<i>Выберите верный ответ.</i> К какому результату приведет выполнение запроса DROP DATABASE Users? 1. Полное удаление базы данных «Users» 2. Блокировка на внесение изменений в базу данных «Users» 3. Удаление таблицы «Users» из текущей базы данных	1	1-3
14.		<i>Выберите верный ответ.</i> Определите оператор, позволяю-	1	1-3

№ п/п	Тип задания	Формулировка задания	Правильный ответ	Время выполнения (в минутах)
		щий удалить таблицу «Студент». 1. DROP TABLE «Студент» 2. DELETE FROM «Студент» 3. DELETE (Студент)		
15.		<i>Выберите верный ответ.</i> Какое выражение используется для возврата неповторяющихся значений? 1. SELECT DISINCT 2. SELECT DIFFERENT 3. SELECT UNIQUE 4. SELECT ALL	1	1-3
16.		Запрос «SELECT surname ____ Students WHERE age ____ 20 AND 22» возвращает фамилии студентов, возраст которых от 20 до 22 лет. <i>Заполните пропущенные места в запросе.</i> 1. INTO, IN 2. FROM, BETWEEN 3. FROM, INTO 4. FROM, LIKE	2	1-3
17.	Задание открытого типа	Напишите определение понятия «сетевая модель».	Сетевая модель – это модель данных позволяет организовывать БД, структура которых представляется графом общего вида.	3-5
18.		Напишите определение понятия «иерархическая модель».	Иерархическая модель – это модель данных позволяет строить БД с иерархической древовидной структурой.	3-5
19.		Напишите определение понятия «объектно-реляционная модель».	Объектно-реляционная модель -> модель данных реализована с помощью реляционных таблиц, но включает объекты, аналогичные объектам в объектно-ориентированном программировании.	3-5
20.		Напишите определение понятия «реляционная модель»?	Реляционная модель - модель данных основана на представлении данных в виде набора отношений, каждое из которых является подмножеством декартова произведения определённых множеств.	3-5
21.		Напишите определение понятия «внешний ключ».	Внешний ключ в реляционной модели данных – атрибут подчинённого (дочернего) отношения,	3-5

№ п/п	Тип задания	Формулировка задания	Правильный ответ	Время выполнения (в минутах)
			который является копией первичного или уникального ключа родительского отношения.	
ПК-8. Способность понимать, совершенствовать и применять современный математический аппарат.				
22.	Задание закрытого типа	<i>Выберите верный ответ.</i> Какая модель данных представляет собой совокупность данных, состоящая из набора двумерных таблиц? 1. сетевая модель 2. иерархическая модель 3. реляционная модель 4. объектно-ориентированная модель	3	1-3
23.		<i>Вставьте пропущенные/ое слова/о.</i> ... – наименьшая поименованная единица данных, к которой СУБД может обращаться непосредственно и с помощью которой выполняется построение всех остальных структур. Для каждого элемента данных должен быть определён его тип 1. запись 2. агрегат данных 3. элемент данных 4. модель данных	3	1-3
24.		<i>Выберите верный ответ.</i> Какая модель данных позволяет организовывать БД, структура которых представляется графом общего вида? 1. сетевая модель 2. иерархическая модель 3. реляционная модель 4. объектно-ориентированная модель	1	1-3
25.		<i>Выберите верный ответ.</i> Какая модель данных позволяет организовывать БД, структура которых представляется графом общего вида? 1. сетевая модель 2. иерархическая модель 3. реляционная модель 4. объектно-ориентированная модель	1	1-3
26.		<i>Выберите верный ответ.</i> Что является основным недостатком иерархической модели данных?	1	1-3

№ п/п	Тип задания	Формулировка задания	Правильный ответ	Время выполнения (в минутах)
		1. дублирование данных 2. зависимость данных от программ 3. актуализация данных 4. синхронизация данных		
27.	Задание открытого типа	Для чего используется ключевое слово HAVING в команде Select?	Ключевое слово HAVING позволяет указать условия выбора для групп записей.	1-3
28.		К какому результату приведет выполнение запроса DROP DATABASE Students?	Выполнение данного запроса приведет к полному удалению базы данных «Students»	1-3
29.		Запрос «SELECT fio ____ Students WHERE age ____ 18 AND 20» возвращает имена студентов, возраст которых от 18 до 20 лет. <i>Запишите запрос, заполнив пропущенные места в запросе.</i>	SELECT fio from Students WHERE age between 18 AND 20	3-5
30.		Напишите запрос, который выводит количество записей в таблице «Students».	SELECT COUNT(*) FROM Students	3-5
31.		Для чего в запросах SQL используется предикат Like?	Предикат Like используется для сравнения строк, применяется только к полям типа CHAR, VARCHAR	3-5

7.4. Методические материалы, определяющие процедуры оценивания результатов обучения по дисциплине (модулю)

Контроль успеваемости по дисциплине осуществляется с помощью следующих оценочных средств:

Таблица 10 – Технологическая карта рейтинговых баллов по дисциплине (модулю)

№ п/п	Контролируемые мероприятия	Количество мероприятий / баллы	Максимальное количество баллов	Срок представления
Основной блок				
1.	<i>Выполнение лабораторных работ</i>	6/7	42	Сроки указаны в Moodle
Всего			42	-
Блок бонусов				
2.	<i>Посещение занятий</i>		8	
Всего			8	-
Дополнительный блок**				
3.	<i>Экзамен</i>			
Всего			50	-
ИТОГО			100	-

Таблица 11. Шкала перевода рейтинговых баллов в итоговую оценку за семестр по дисциплине

Сумма баллов	Оценка по 4-балльной шкале
90–100	5 (отлично)
85–89	4 (хорошо)
75–84	
70–74	
65–69	
60–64	3 (удовлетворительно)
Ниже 60	2 (неудовлетворительно)

При реализации дисциплины в зависимости от уровня подготовленности обучающихся могут быть использованы иные формы, методы контроля и оценочные средства, исходя из конкретной ситуации.

8. УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

8.1. Основная литература

1. Hopcroft J. E., Motwani R., Ullman J. D. Introduction to Automata Theory, Languages, and Computation (3rd Edition). — Addison-Wesley, Boston, MA, USA, 2006. — 750 с.
2. Шень А. Программирование: теоремы и задачи. — М.: МЦНМО, 2014. — 296 с.
3. Шень А., Верещагин Н. Языки и исчисления. — М.: МЦНМО, 2012. — 240 с.
4. Верещагин, Н. К. Колмогоровская сложность и алгоритмическая случайность [Электронный ресурс] / Н. К. Верещагин, В. А. Успенский, А. Шень. — Электрон. дан. — СПб: Лань, 2013. — 575 с. — Режим доступа: <https://e.lanbook.com/book/56395> — Загл. с экрана.

8.2. Учебно-методическое обеспечение для самостоятельной работы обучающихся

1. Кривцова, И. Е. Основы дискретной математики. Часть 1. Учебное пособие [Электронный ресурс] / И. Е. Кривцова, И. С. Лебедев, А. В. Настека. — Электрон. дан. — СПб: ИТМО, 2016. — 92 с. — Режим доступа: http://books.ifmo.ru/book/1869/osnovy_diskretnoy_matematiki_chast_1_uchebnoe_posobie.htm — Загл. с экрана.
2. Демченко, К. А. Базы данных : учебное пособие / К. А. Демченко. — Чита : ЗабГУ, 2023. — 121 с. — ISBN 978-5-9293-3301-9. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/438206>

8.3. Дополнительная литература

1. Вики-конспекты. — http://neerc.ifmo.ru/wiki/index.php?title=Заглавная_страница

8.4. Интернет-ресурсы, необходимые для освоения дисциплины (модуля)

Электронный каталог Научной библиотеки АГУ на базе MARK SQL НПО «Информ-систем»: <https://library.asu.edu.ru>

1. Корпоративный проект Ассоциации региональных библиотечных консорциумов (АРБИКОН) «Межрегиональная аналитическая роспись статей» (МАРС): <http://mars.arbicon.ru>
2. Единое окно доступа к образовательным ресурсам <http://window.edu.ru>

9. МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

Для проведения лекционных занятий используется аудитория, оборудованная современной презентационной техникой (проектор, экран, ноутбук).

Для выполнения лабораторных работ используются компьютерные классы с установленным в них необходимым программным обеспечением.

Рабочая программа дисциплины (модуля) при необходимости может быть адаптирована для обучения (в том числе с применением дистанционных образовательных технологий) лиц с ограниченными возможностями здоровья, инвалидов. Для этого требуется заявление обучающихся, являющихся лицами с ограниченными возможностями здоровья, инвалидами, или их законных представителей и рекомендации психолого-медико-педагогической комиссии. Для инвалидов содержание рабочей программы дисциплины (модуля) может определяться также в соответствии с индивидуальной программой реабилитации инвалида (при наличии).