Автономная некоммерческая образовательная организация
высшего образования Центросоюза Российской Федерации
«Сибирский университет потребительской кооперации»

Методические указания и задания по выполнению
практических, лабораторных и самостоятельных работ
по профессиональному модулю

ПМ.02 АДМИНИСТРИРОВАНИЕ БАЗ ДАННЫХ

по специальности
09.02.13 Интеграция решений с применением технологий искусственного интеллекта

(направленность программы: Применение искусственного интеллекта)

квалификация выпускника:
Специалист по работе с искусственным интеллектом

Новосибирск
2025

Методические указания и задания по выполнению практических, лабораторных и самостоятельных работ по профессиональному модулю *«Администрирование баз данных»* для обучающихся среднего профессионального образования по специальности *09.02.13* Интеграция решений с применением технологий искусственного интеллекта/ [сост.: Е.В. Бабанова, ст. преподаватель /− Новосибирск, 2025.

Рецензент:
Лихачев В.В., канд. техн. наук, доцент кафедры информатики.

Методические указания и задания утверждены и рекомендованы к использованию в учебном процессе кафедрой информатики, протокол от 28 мая 2025 г. № 9.

СОДЕРЖАНИЕ

1. ОБЩИЕ ПОЛОЖЕНИЯ

Методические указания и задания к лабораторным занятиям и самостоятельной работе по профессиональному модулю *ПМ.02 Администрирование баз данных* разработаны для обучающихся специальности *09.02.13 Интеграция решений с применением технологий искусственного интеллекта*.

Освоение профессионального модуля способствует подготовке выпускника к решению профессиональных задач соадминистрирования баз данных и серверов в соответствии специальности *09.02.13 Интеграция решений с применением технологий искусственного интеллекта*

– соадминистрирование серверов и баз данных;

– разработка политики безопасности SQL сервера, базы данных и отдельных объектов базы данных;

– применение законодательства Российской Федерации в области сертификации программных средств информационных технологий;

– проектирование и создание базы данных;

– выполнение запросы по обработке данных на языке SQL;

– владение технологиями проведения сертификации программного средства.

2. ТЕМЫ И ИХ КРАТКОЕ СОДЕРЖАНИЕ

Тема 1. Принципы построения и администрирования баз данных

Предметная область. Описание предметной области. Структурирование. Объект. Класс объектов. Связи между объектами.

Информационная модель данных. Последовательность создания информационной модели.

Понятие нормализации отношений. Первая нормальная форма. Вторая нормальная форма. Функциональная зависимость. Третья нормальная форма. Транзитивная зависимость.

Типы связей: один к одному, один ко многим, многие ко многим. Реляционная модель данных. Основные термины: отношение, кортеж, тип данных, домен, атрибут, таблица, запись, связь, запрос. Проектирование данных. Компоненты и бизнес процессы. Система данных. Задача и система задач. Пример формализованного описания задачи.

Тема 2. Серверы баз данных

Понятие сервера. Классификация серверов. Типы серверов: файловые, печати, приложений, сообщений, баз данных.

Базовые сетевые топологии и комбинированные топологические решения. Достоинства и недостатки базовых сетевых топологий. Архитектура «клиент–сервер». Принципы разделения между клиентскими и серверными частями. Типовое разделение функций.

Модели данных, основные операции и ограничения; технологию установки и настройки сервера баз данных; требования к безопасности сервера базы данных; государственные стандарты и требования к обслуживанию баз данных.

Тема 3. Администрирование баз данных и серверов

Соадминистрирование серверов; разработке политики безопасности SQL сервера, базы данных и отдельных объектов базы данных; применении законодательства Российской Федерации в области сертификации программных средств информационных технологий.

Достоинства и недостатки языка SQL. Стандарты языка SQL. Возможности современного языка SQL. Общая структура директивы SELECT. Однотабличные запросы на выборку данных. Выборка записей по заданному условию. Сортировка результатов запроса. Встроенные функции языка SQL. Запросы с группировкой записей. Групповые (агрегатные) функции. Использование вложенных запросов (подзапросов). Многотабличные запросы с операцией JOIN. Работа с результатами нескольких запросов. Запросы на изменение данных в таблицах БД (операторы UPDATE и DELETE). Использование оператора MERGE.

3. МЕТОДИЧЕСКИЕ УКАЗАНИЯ И ЗАДАНИЯ К ЗАНЯТИЯМ ЛАБОРАТОРНОГО ТИПА И САМОСТОЯТЕЛЬНОЙ РАБОТЕ

Самостоятельная работа является одним из видов учебной деятельности обучающихся, способствует развитию самостоятельности, ответственности и организованности, творческого подхода к решению проблем учебного и профессионального уровня. Самостоятельная работа проводится с целью:

– систематизации и закрепления полученных теоретических знаний и практических умений обучающихся;

– углубления и расширения теоретических знаний;

– формирования умений использовать специальную литературу;

– развития познавательных способностей и активности обучающихся, творческой инициативы, ответственности и организованности;

– формирования самостоятельности мышления, способностей к саморазвитию, самосовершенствованию и самореализации.

Аудиторная самостоятельная работа по учебной дисциплине проводится на учебных занятиях под непосредственным руководством преподавателя и по его заданию (сквозная задача приведена ниже). Внеаудиторная самостоятельная работа выполняется по заданию преподавателя без его непосредственного участия для обучающихся очной форм обучения.

Основными видами аудиторной самостоятельной работы являются:

– обсуждение теоретических вопросов и решение практических задач по темам дисциплины;

– работа с литературой и другими источниками информации, в том числе электронными.

Решение сквозной задачи осуществляется на лабораторных занятиях в соответствии с графиком учебного процесса. Для подготовки к лабораторным занятиям обучающиеся должны изучить соответствующую главу основного учебного пособия, рекомендованную специальную литературу. Кроме того, необходим самостоятельный поиск и подбор литературы, включая монографии и журнальные публикации, информацию из сети Интернет.

Внеаудиторная самостоятельная работа выполняется по заданию преподавателя, но без его непосредственного участия. Перед выполнением внеаудиторной самостоятельной работы преподаватель проводит консультацию с определением цели задания, его содержания, сроков выполнения, ориентировочного объема работы, основных требований к результатам работы, критериев оценки, форм контроля и перечня литературы. В процессе консультации преподаватель предупреждает о возможных типичных ошибках, встречающихся при выполнении задания.

Видами заданий для внеаудиторной самостоятельной работы могут быть:

– подготовка сообщений к выступлению на семинаре, конференции; подготовка рефератов, докладов;

– решение задач.

При выполнении внеаудиторной самостоятельной работы обучающийся имеет право обращаться к преподавателю за консультацией с целью уточнения задания, формы контроля выполненного задания. Контроль результатов внеаудиторной самостоятельной работы обучающихся может проводиться в письменной, устной или смешанной форме.

Лабораторное занятие № 1. «Построение схемы базы данных»

Постановка задачи. В процессе выполнения лабораторных работ будет использована модель реальной задачи, которая связана с управлением товарооборотом на складе. Общая задача состоит в создании реляционной базы данных для соответствующей совокупности информационных объектов (ИО).

База данных состоит из следующих таблиц: 1) Товары; 2) Поставщики; 3) Заказы; 4) Клиенты.

Структура таблиц базы данных

Таблица	Названия столбцов	Тип данных
Товары	Код товара	Числовой
	Группа товара	Текстовый
	Наименование товара	Текстовый
	Категория	Текстовый
	Код поставщика	Числовой
	Дата поступления	Дата/Время
	Закупочная цена	Числовой
	Остаток товара на складе	Числовой
Поставщики	Код поставщика	Числовой
	Название	Текстовый
	Город	Текстовый
	E-mail	Текстовый
	Специализация	Текстовый
Заказы	Код заказа	Числовой
	Код клиента	Числовой
	Код товара	Числовой
	Дата заказа	Дата/Время
	Количество	Числовой
Клиенты	Код клиента	Числовой
	Наименование фирмы	Текстовый
	Телефон	Текстовый
	ФИО	Текстовый
	Фотография	Вложение

Порядок выполнения лабораторной работы

Задание 1. Создать файл базы данных

Задание 2. Создать таблицы БД в соответствии с заданной структурой и индивидуальными свойствами отдельных полей
Задание 3. Используя источники данных, указанные преподавателем, заполнить таблицы конкретными данными
Задание 4. Установить связи между таблицами и получить требуемую схему данных.

Лабораторное занятие № 2. «Составление словаря данных»

В СУБД Microsoft Access имеются широкие возможности по конструированию графического интерфейса пользователя для удобной работы с данными. Одним из важнейших инструментов работы являются экранные формы ввода/вывода, которые позволяют осуществлять первоначальную загрузку записей в таблицы БД, выполнять их просмотр, а также производить корректировку данных, т.е. добавлять и удалять записи, изменять значения в конкретных полях. При этом содержимое БД отображается через форму в привычном для пользователя виде – как правило, в виде первичного документа.

При наличии схемы данных, состоящей из нескольких взаимосвязанных таблиц, могут быть созданы сложные экранные формы, обеспечивающие корректную работу с данными, которые характеризуются наличием логических связей. Такие формы в значительной степени соответствуют макетам первичных документов со сложной многоуровневой организацией.

При разработке экранных форм, которые будут обеспечивать загрузку взаимосвязанных таблиц БД, следует придерживаться определенных требований к последовательности их загрузки в соответствии со схемой данных. Эти требования можно сформулировать следующим образом:

- независимо могут загружаться таблицы, которые не подчиняются каким-либо другим таблицам;
- подчиненные таблицы могут загружаться либо одновременно с главными таблицами, либо после загрузки данных в главные таблицы (в противном случае возникают нарушения целостности связей между главными и подчиненными записями);
- в базу данных сначала загружаются справочные данные, а затем – учетные.

Порядок выполнения лабораторной работы

Задание 1. Построить ***простые (однотабличные)*** формы и произвести их модификацию в режиме Конструктора (форматирование элементов формы, добавление новых элементов управления и т.п.)
Задание 2. Построить ***сложные (составные, многотабличные)*** формы и произвести их модификацию в режиме Конструктора (форматирование элементов формы, добавление новых элементов управления и т.п.)

Лабораторное занятие № 3. «Разработка технических требований к серверу баз данных»

При большем количестве пользователей требуется добавлять дополнительные сервера приложений. Для построения данной системы лучше использовать виртуализацию серверов с единым хранилищем данных.

Так же, на сервере, необходим выход в интернет, для проверки подлинности ПО, которая проходит каждый раз при авторизации сотрудника в комплексе.

В случае использования архитектуры с бездисковыми рабочими станциями (терминальный доступ) следует увеличить требуемый объем памяти сервера приложений на величину, рассчитываемую по формуле:500 Мб * число клиентов.

Обязательно наличие у сервера и сетевых концентраторов источников бесперебойного питания (ИБП) для исключения разрушения базы данных в случае аварийного выключения питания.

Следует учитывать, что приведенные выше требования достаточны для комфортной работы. Использование требований, указанных в следующей строке, безусловно, повысит производительность. Не запрещается в случае финансовых сложностей использование на этапе внедрения техники согласно предыдущей строке, но нужно планировать в течение не более года обновление, т.к. по мере увеличения объема базы будет падать быстродействие. Также могут меняться требования к объему свободного пространства на жестком диске — они будут возрастать с увеличением объема вашей базы данных.

Лабораторное занятие № 4. «Разработка требований к корпоративной сети»

На сервере при числе рабочих мест до 1-6 и объеме базы данных до 10 Гб в качестве СУБД допустимо использовать бесплатную версию MS SQL 2017 Express Edition (10 Гб — это максимальный объем, поддерживаемый данной версией СУБД), а потом в качестве ОС допустимы MS Windows 8 / Windows 10 (также допустимы серверные ОС: Windows server 2012). Если в системе действует брандмауэр, то необходимо открыть в нем порт 1433.

В случае использования архитектуры с терминальным доступом на сервере должна быть установлена операционная система, поддерживающая терминальные подключения (Windows Server 2012 с установленной службой Windows Terminal Service). Недопустимо использование Small Business Server (эта ОС не поддерживает терминальные подключения).

В случае использования архитектуры «Тонкий клиент» на клиентских компьютерах может быть установлена операционная система семейства Linux GTK.

Лабораторное занятие № 5. «Конфигурирование сети»

Прежде чем конфигурировать TCP/IP, нужно узнать определенную информацию о параметрах сети. В большинстве случаев необходимую информацию предоставит системный администратор. Потребуются следующие параметры.

- IP-адрес. Это уникальный адрес компьютера в формате четырех трехзначных десятичных чисел, разделенных точками, например, 128.253.153.54. Этот набор чисел должен предоставить системный администратор.

  Если конфигурируется только режим работы с заглушкой (loopback mode), т.е. когда нет подключения SLIP и нет адаптера Ethernet, а только моделируется подключение TCP/IP компьютера к самому себе, то IP-адресом будет 127.0.0.1.

- Маска имен в сети ("netmask"). Формат маски аналогичен формату IP-адреса. Маска определяет, какая часть IP-адреса соответствует номеру локальной (под)сети (subnetwork number), а какая -- номеру компьютера в сети. Маска представляет собой битовый шаблон. При наложении этого шаблона на адрес компьютера (хоста) можно узнать номер того участка сети, к которому относится этот адрес. Это очень важно для рассылки сообщений, поэтому если вдруг окажется, например, что можно установить связь с кем-либо вне локальной сети, а внутри сети такая связь не устанавливается, то может оказаться, что маска указана неверно.

  Маска выбиралась администратором сети при ее проектировании, поэтому он должен ее точно знать. Большинство локальных сетей имеют класс С и используют маску 255.255.255.0. Сети класса В имеют маску 255.255.0.0. Программа NET-3 выбирает маску автоматически, предполагая по умолчанию, что подсети отсутствуют, если они не указаны.

  То же относится и к адресу заглушки. Поскольку адрес в этом случае всегда 127.0.0.1, то и маска всегда 255.0.0.0. Это можно указать явно или положиться на маску, даваемую по умолчанию.

- Адрес сети (network address). Это адрес является результатом побитовой операции "И" двух аргументов: IP-адреса компьютера и маски. Например, если маска имеет вид 255.255.255.0, IP-адрес равняется 128.253.154.32, то адрес сети будет 128.253.154.0. Если маска равняется 255.255.0.0, то адрес сети будет равняться 128.253.0.0.

  При использовании только заглушки сетевой адрес отсутствует.

- Широковещательный адрес (broadcast address). Этот адрес используется для трансляции пакетов сообщений по всем компьютерам, объединенным в (под)сеть. Следовательно, если IP-адрес компьютера в сети определяется последним байтом (иными словами, если маска равна 255.255.255.0), то широковещательный адрес является результатом побитовой операции "ИЛИ" выражения 0.0.0.255 с IP-адресом компьютера. Например, если IP-адрес компьютера 128.253.154.32, а маска 255.255.255.0, то широковещательный адрес будет равен 128.253.154.255.

Исторически сложилось, что в некоторых сетях в качестве широковещательного используется адрес сети. Этот факт следует знать, и если по этому вопросу возникают сомнения, надо обращаться к системному администратору. Разумеется, во многих случаях для конфигурирования сети достаточно будет взять соответствующие файлы с других компьютеров в той же сети и изменить в них только IP-адрес компьютера.

При использовании только заглушки широковещательный адрес отсутствует.

- Адрес шлюза (gateway address). Это адрес компьютера, который является действительно "шлюзом" во внешний мир (т.е. к компьютерам вне локальной сети). Во многих случаях адрес шлюза получается из IP-адреса компьютера заменой последних групп цифр на ".1". Например, если IP-адрес компьютера равняется 128.253.154.32, то адрес шлюза может иметь значение 128.253.154.1. Это значение может также сообщить системный администратор.

Реально шлюзов может быть несколько. В действительности *шлюз* -- это компьютер, который находится в двух различных сетях (т.е. имеет IP-адреса в различных подсетях). Шлюз занимается тем, что пересылает пакеты сообщений между этими двумя сетями. Многие сети имеют единственный шлюз во "внешний мир" (который представляет собой непосредственно примыкающую сеть). Однако в некоторых случаях к локальной сети примыкает несколько других сетей, и для каждой из них имеется свой шлюз.

При использовании только заглушки адрес шлюза отсутствует. То же верно и в случае, когда локальная сеть изолирована от всех остальных.

- Адрес сервера имен в сети (name server address). У большинства компьютеров в сети имеется сервер, который преобразует имя компьютера в IP-адрес. Адрес этого сервера должен сообщить администратор сети. Такой сервер можно также запустить на собственном компьютере (программа называется `named`); в этом случае адрес сервера имен в сети равняется 127.0.0.1. Кроме тех

случаев, когда *абсолютно необходимо* иметь собственный сервер имен, лучше воспользоваться тем, который имеется (если имеется). Совершенно отдельный вопрос -- конфигурирование программы `named`. Сейчас он рассматриваться не будет, поскольку цель данного раздела -- дать возможность приступить к работе в сети. Вопросы, касающиеся имен в сети, будут рассмотрены позднее.

При использовании только заглушки адрес сервера имен отсутствует.

Пользователям SLIP/PPP информация, приведенная выше (кроме адреса сервера имен), может не требоваться. При подключении SLIP клиентский IP-адрес определяется либо "статически", когда при каждом новом соединении клиенту дается один и тот же IP-адрес, либо "динамически", когда адрес в момент соединения выбирается из некоторого множества свободных IP-адресов сервера. В последующих разделах конфигурирование подключения SLIP будет обсуждаться более подробно.

В реализации NET-3 заложена полная маршрутизация (full routing), поддерживаются множественные маршруты и подсети (последние -- на данный момент только по границам байта), и тому подобное. Приведенное выше иллюстрирует только принципиальные основы построения сетей TCP/IP. Конкретная сеть может обладать рядом особенностей; выяснить их можно у администратора сети. Также полезно будет почитать экранную документацию к программам `route` и `ifconfig`. Конфигурирование сетей TCP/IP лежит далеко за пределами темы данной книги, однако приведенное рассмотрение должно дать достаточно информации, чтобы пользователь смог выяснить остальное самостоятельно.

Лабораторное занятие № 6. «Сравнение технических характеристик серверов»

Основные системные требования, которым должен удовлетворять компьютер для успешной инсталляции на нем сервера БД Oracle XE для Microsoft Windows, приведены в таблице 1.

Таблица 1. Требования к аппаратному обеспечению

Компонент аппаратного обеспечения	**Требования**
Процессор	Intel x86
Объем дисковой памяти	Минимальный 1,6 Гбайт Рекомендуется 5 Гбайт
Объем оперативно памяти	Минимальный 256 Мбайт Рекомендуется 512 Мбайт

Лабораторное занятие № 7. «Формирование аппаратных требований и схемы банка данных»

Задание 1.

Для проверки того, что компьютер удовлетворяет системным требованиям для инсталляции СУБД Oracle XE под ОС Microsoft Windows, следует выполнить действия, перечисленные ниже.

- На рабочем столе выбрать: Кнопка Пуск | Программы | Стандартные | Служебные | Сведения о системе.
- В открывшемся окне «Сведения о системе» щелкнуть мышкой на опции Сведения о системе для отображения информации о процессоре компьютера, об объеме физической памяти и о версии ОС Microsoft Windows.
- В окне «Сведения о системе» выбрать опции Компоненты, Сеть, а затем выбрать опцию Протокол и выяснить, доступен ли протокол TCP/IP (есть ли его описание).
- На рабочем столе щелкнуть по ярлыку Мой компьютер. В поле открывшегося окна щелкнуть правой кнопкой мышки и в появившемся контекстном меню выбрать опции Вид | Таблица чтобы определить свободное место на жестких дисках компьютера.
- В окне «Сведения о системе» выбрать опцию Программная среда, а затем выбрать опцию Загруженные модули. В появившемся списке загруженных модулей требуется найти модуль msi и выяснить его версию.
- В окне «Сведения о системе» выбрать опции Параметры обозревателя, Internet Explorer, а затем выбрать опцию Итоги для определения версии Internet Explorer. Если на компьютере используется другой Web-браузер (не Internet Explorer), то его версию в большинстве случаев можно узнать с помощью опции About (о программе) в меню Help (Справка).
- После работы с приложением «Сведения о системе» следует выбрать опции Файл и Выход.

Наиболее простой вариант записи команды SELECT можно представить в виде следующей синтаксической формулы:

SELECT ⟨ Список_выбора ⟩
FROM ⟨ Набор_источников_данных ⟩

В представленной формуле с помощью угловых скобок ⟨…⟩ указаны некоторые синтаксические элементы. Общий смысл этих элементов можно понять из текста, который содержится в скобках, а более подробная расшифровка будет дана в дальнейших пояснениях.

После ключевого слова SELECT нужно определить столбцы, которые должны присутствовать в выходной таблице, а после слова FROM задается перечень таблиц и других источников данных, с которыми должен работать запрос.

Важно понимать, что с помощью указанной конструкции пользователь лишь описывает желаемый набор данных, которые будут представлены в виде выходной таблицы, но явно не указывает детальный план действий по выполнению запроса. Построение такого плана – это задача оптимизатора запросов, который имеется в составе любой СУБД.

Рассмотрим простейший случай полного отображения таблицы:

SELECT * FROM ⟨ имя_исх._таб. ⟩

Здесь символ ***** означает «все столбцы исходной таблицы».

Если требуется отобразить конкретные столбцы таблицы, то список этих столбцов нужно указать после ключевого слова SELECT:

SELECT ⟨ список_столбцов ⟩ FROM ⟨ имя_исх._таб. ⟩

Выводимое имя столбца можно поменять при помощи *псевдонима* (*alias*). В этом случае для элемента списка выбора применяется следующая конструкция (здесь и в дальнейшем квадратными скобками […] обозначаются *необязательные* элементы):

⟨ элемент_списка ⟩ [AS] ⟨ псевдоним ⟩

Если псевдоним содержит пробелы или специальные символы, то его нужно поместить в двойные кавычки.

В списке выбора могут присутствовать выражения языка SQL, что означает включение вычисляемых полей в выходную таблицу. Обычно выражение – это некоторая комбинация полей в таблицах, констант и операторов.

Для **числовых** типов используются **арифметические** операции: **+** , **-** , ***** , **/** . Можно сцеплять элементы типа «строка символов», т.е. выполнять операцию конкатенации (||). При построении сложных выражений разрешается применять круглые скобки.

В качестве имени вычисляемого столбца по умолчанию выводится соответствующая формула. Вместо этого гораздо удобнее использовать псевдоним.

Если в начале списка выбора указать служебное слово **DISTINCT**, то из

результата выборки будут **исключены повторяющиеся строки**. **Важно** помнить, что слово DISTINCT относится ко **всему списку выбора**, т.е. его указывают **только один раз в начале этого списка**.

При выборке записей по заданному условию необходимо добавить раздел WHERE в состав директивы SELECT:

SELECT ⟨ список_столбцов ⟩
FROM ⟨ имя_исх._таб.⟩
WHERE ⟨ условие_отбора ⟩

Условие отбора часто включает в себя простые операции сравнения: = , < , > , <= , >=, < > .

Наряду с этим, в условии отбора может присутствовать оператор **LIKE '**⟨шаблон⟩**'**, который выявляет в текстовом поле наличие **подстроки**, задаваемой в виде шаблона. Шаблон (маска) поиска может включать в себя следующие **специальные символы**:

- **_** – один произвольный символ;
- **%** – последовательность любых символов (в том числе ни одного).

Попадание в **замкнутый** числовой интервал [a, b] проверяется с помощью условного оператора **BETWEEN** a **AND** b

Принадлежность к множеству (списку) значений, которые заданы в явном виде, можно проверить с помощью условного оператора **IN (** ⟨список⟩ **)**. **Важно** отметить, что элементы списка сами могут быть списками (множествами) значений. Например:

SELECT department_id, last_name, salary
FROM employees
WHERE (department_id, salary) IN ((20, salary), (60, 4800), (90, 17000));

Отсутствующие (пустые) значения (**NULL**) имеют ряд **особенностей** при выполнении с ними обычных операций. В упрощенном виде можно полагать, что любая операция с NULL всегда дает NULL. В частности, если значение X неизвестно (NULL), то сравнение X=X **не дает** значения «истина».

Проверку на отсутствие значения (т. е. наличие NULL) выполняет специальный оператор **IS**. Например*:*

SELECT last_name **FROM** employees
WHERE commission_pct **IS NULL**

При построении сложных (составных) условий применяют логические операции **AND**, **OR** и **NOT**.

Чтобы обеспечить сортировку результатов запроса, нужно добавить раздел ORDER BY:

SELECT ⟨ список_выбора ⟩
FROM ⟨ имя_исх._таб. ⟩
ORDER BY ⟨ Ключи_сортировки ⟩

Ключ сортировки обычно включает в себя имя столбца (или псевдоним) из списка выбора, либо вместо этого можно указать порядковый номер элемента из списка SELECT. Порядок сортировки (для каждого ключа)

определяют следующие указатели:

- ASC – возрастающий порядок (по умолчанию);
- DESC – убывающий порядок.

При наличии нескольких ключей сортировки, как обычно, каждый следующий ключ начинает действовать в случае одинаковых значений всех предыдущих ключей.

Встроенные функции. Язык SQL в СУБД Oracle располагает большим количеством встроенных (стандартных) функций. Встроенная функция обычно имеет некоторое количество входных аргументов, с учетом которых выдается результирующее значение.

Встроенные функции в СУБД Oracle разделены на несколько категорий:

- символьные функции – манипулируют со строками символов;
- числовые функции – выполняют расчеты с числовыми данными;
- функции для работы с датой и временем;
- функции преобразования типов данных и др.

Рассмотрим некоторые из перечисленных функций:

- функция **SUBSTR** (⟨str⟩, m [, n]) выделяет из исходной строки **⟨str⟩** подстроку, которая начинается с позиции **m** и имеет длину **n** символов;
- функция **UPPER** (⟨str⟩) возвращает представление исходной строки **⟨str⟩** с использованием только заглавных букв:
- функция **LENGTH** (⟨str⟩) возвращает длину (число знаков) исходной строки **⟨str⟩**;
- функция **MONTHS_BETWEEN** (⟨d2⟩, ⟨d1⟩) находит количество месяцев между датами **⟨d1⟩** (начало интервала) и **⟨d2⟩** (конец интервала);
- функция **SYSDATE** – возвращает текущую дату (аргументы отсутствуют);
- функция **TO_CHAR** (⟨expr⟩ [, ⟨fmt⟩]) преобразует в символьную строку результат, который получен с помощью выражения **⟨expr⟩** и имеет числовое значение (или дата/время); преобразование осуществляется с учетом формата **⟨fmt⟩**;
- функция **NVL** (⟨expr1⟩ , ⟨expr2⟩) возвращает выражение **⟨expr2⟩**, если выражение **⟨expr1⟩** имеет неопределенное значение (NULL).

Функция **DECODE** производит условную замену конкретных величин. При этом в рамках директивы языка SQL реализуется логика IF-THEN-ELSE. Синтаксис обращения к функции выглядит следующим образом:

DECODE (⟨expr⟩**,** ⟨s_val1⟩**,** ⟨res1⟩
[, ⟨s_val2⟩**,** ⟨res2⟩
[, …]]
[, ⟨res_def⟩ **])**

Функция последовательно сравнивает значение **⟨expr⟩** со значениями **⟨s_val1⟩, ⟨s_val2⟩** и т. д. При первом же совпадении дальнейшие сравнения прекращаются, и функция возвратит значение **⟨res⟩** с соответствующим номером. Если ни одного совпадения не произошло, то результатом будет **⟨res_def⟩**, а если этот элемент не указан, то функция возвратит NULL.

Очень часто при **статистической обработке** данных используются **агрегатные** функции:

Функция	Результат
COUNT( )	количество значений
SUM( )	сумма значений
AVG( )	среднее значение
MIN( )	минимальное значение
MAX( )	максимальное значение

В общем случае аргументом **агрегатной** функции является обычное выражение, которое будет вычисляться при обработке записей. В аргументе функции можно указать служебное слово DISTINCT, чтобы учесть только несовпадающие значения. Для функции COUNT разрешается следующий вариант обращения к этой функции: COUNT(*****).

Варианты заданий

Задание 1. Выбрать из таблицы EMPLOYEES столбцы FIRST_NAME, LAST_NAME, HIRE_DATE.

Задание 2. Сделать выборку столбцов из таблицы EMPLOYEES с применением псевдонимов.

Задание 3. Используя параметр месячная зарплата сотрудника (столбец SALARY в таблице EMPLOYEES), вычислить годовой доход для каждого.

Задание 4. При выборке данных из таблицы EMPLOYEES значения полей FIRST_NAME и LAST_NAME для каждого сотрудника поместить в один столбец с названием FULL_NAME.

Задание 5. По данным таблицы EMPLOYEES выдать неповторяющиеся значения JOB_ID.

Задание 6. По данным таблицы EMPLOYEES неповторяющиеся значения JOB_ID выдать только в пределах отдельного департамента.

Задание 7. По таблице EMPLOYEES получить список сотрудников с низкой зарплатой (не более 2500$).

Задание 8. По таблице EMPLOYEES получить список сотрудников, у которых в фамилии (LAST_NAME) два предпоследних символа – ‘en’.

Задание 9. По таблице EMPLOYEES получить список сотрудников, у которых зарплата находится в диапазоне [3000$, 7000$].

Задание 10. По таблице EMPLOYEES получить список сотрудников, которые работают в департаментах 20, 60 и 90.

Задание 11. По таблице EMPLOYEES получить списки сотрудников:

- для категории клерков (поле job_id содержит подстроку CLERK), у которых зарплата находится в диапазоне [2500$, 3000$];
- которые приняты на работу с начала 1999 г. и не относятся к департаментам 50, 80, 100.

Задание 12. По таблице EMPLOYEES получить список сотрудников, у которых зарплата не ниже 10 000$, с указанием кода подразделения

(department_id) и фамилии (last_name). Список отсортировать сначала по коду подразделения, а в пределах одного подразделения – по фамилии.

Задание 13. По таблице EMPLOYEES сформировать список сотрудников, у которых первая буква фамилии (last_name) находится в интервале от ‘F’ до ‘K’. Также для каждого сотрудника получить **идентификатор**, который объединяет 3 первых символа имени (first_name) и 2 первых символа фамилии (в виде **заглавных букв**).

Задание 14. По таблице EMPLOYEES получить список сотрудников с указанием фамилии (last_name) и инициала – первая буква имени (first_name) с точкой. При выводе сортировать список по убыванию количества букв в фамилии, а фамилии с одинаковым количеством букв – по алфавиту.

Задание 15. По таблице EMPLOYEES получить список сотрудников, где указать фамилию и имя (last_name, first_name), текущий оклад (salary) и дату приема на работу (hire_date). Одновременно для каждого сотрудника определить стаж работы (в месяцах) и начислить бонус в размере 1 % от оклада за каждый месяц работы. При выводе сортировать список по убыванию стажа работы.

Задание 16. С помощью числового формата (например, ‘999.9’) и аналогичного денежного формата (‘$999.99’) убрать лишние знаки при выводе стажа работы и бонуса в запросе из предыдущего задания.

Задание 17. По данным из таблицы EMPLOYEES посчитать месячную зарплату сотрудников с учетом комиссионной надбавки (поле commission_pct). Результат упорядочить по убыванию размера зарплаты.

Задание 18. По таблице EMPLOYEES получить список сотрудников, где указать фамилию и имя, а также код должности (job_id). При этом в столбце job_id значения SA_REP и SA_MAN нужно заменить строками ‘Торговый представитель’ или ‘Менеджер по продажам’, соответственно. Для остальных значений job_id нужно выводить строку ‘Другое’.

Задание 19. Для сотрудников, стаж работы которых в компании не превышает 15 лет, с помощью запроса по таблице EMPLOYEES найти:

- их число;
- минимальное, максимальное и среднее значение по окладу (SALARY);
- суммарную месячную зарплату с учетом комиссионной надбавки.

Лабораторное занятие № 9. «Установка и настройка сервера под UNIX»

Часто при работе с табличными данными нужно выполнить их **группировку**, т.е. сделать так, чтобы в одну группу попадали записи с одинаковыми значениями для заданных атрибутов (**ключи группировки**). В этом случае применяется следующая команда:

SELECT ⟨ список_столбцов ⟩
FROM ⟨ имя_исх._таб. ⟩
GROUP BY ⟨ Ключи_группировки ⟩

Важно отметить, что логика работы запросов с группировкой требует

наличия жесткой связи между разделами **SELECT** и **GROUP BY.** В частности, любой элемент списка выбора в разделе **SELECT** должен иметь единственное значение для каждой группы. Следовательно, в этом списке могут быть только:

- имена столбцов, которые являются ключами группировки;
- агрегатные функции;
- выражения, состоящие из перечисленных выше элементов.

Группировка по **нескольким** ключам позволяет **детализировать** итоговые результаты.

Для отбора определенных групп по некоторому условию служит раздел **HAVING**. Это происходит по аналогии с разделом **WHERE,** когда идет отбор определенных записей. Раздел **HAVING** может применяться только вместе с разделом **GROUP BY**.

Например, по таблице EMPLOYEES получим список департаментов, в которых число сотрудников больше 5:

```
SELECT  DEPARTMENT_ID, COUNT(*)
FROM   EMPLOYEES
GROUP BY   DEPARTMENT_ID
HAVING COUNT(*) > 5
```

Варианты заданий

Задание 1. По таблице EMPLOYEES найти минимальный, максимальный и средний оклад (SALARY) для каждого департамента.

Задание 2. Для каждого департамента сведения о минимальном, максимальном и среднем окладе требуется получить **по отдельным должностям**.

Задание 3. Получить список департаментов, в которых число сотрудников с низкой зарплатой (меньше 3000$) превышает 3.

Лабораторное занятие № 10. «Выполнение запросов к базе данных»

В разделе **WHERE** оператора **SELECT** может присутствовать **вложенный запрос**. Результат выполнения этого внутреннего запроса (подзапроса) передается внешнему запросу.

Рассмотрим сначала **скалярный** подзапрос, который возвращает **единственное** значение. Например, получим список сотрудников, у которых оклад выше среднего:

```
SELECT   FIRST_NAME, LAST_NAME
FROM   EMPLOYEES
WHERE   SALARY >
        ( SELECT  AVG (SALARY)  FROM EMPLOYEES  )
```

Скалярный подзапрос может быть не только частью логического условия в разделе **WHERE**. Например, воспользуемся скалярным подзапросом в **арифметическом выражении** из раздела **SELECT**, чтобы в списке сотрудников, которые имеют оклад выше среднего, получить также коэффициент превышения:

```
SELECT  FIRST_NAME, LAST_NAME ,
            SALARY / ( SELECT  AVG (SALARY)
                              FROM  EMPLOYEES )  AS  Koeff
FROM   EMPLOYEES
WHERE   SALARY >
                    ( SELECT  AVG (SALARY)  FROM EMPLOYEES  )
```

С помощью подзапроса можно получить **список**, который затем передается внешнему запросу. Например, получим список сотрудников, которые относятся к службам Marketing, Sales и IT:

```
SELECT   FIRST_NAME, LAST_NAME
FROM   EMPLOYEES
WHERE  department_id  IN
              ( SELECT  department_id
              FROM  DEPARTMENTS
              WHERE  department_name  IN('Marketing', 'Sales', 'IT') )
```

Список, который формирует вложенный подзапрос, может содержать **составные элементы** (например, пары значений). С помощью такой конструкции получим список, в который из каждого департамента должны попадать только сотрудники с самым высоким окладом (**элита**):

```
SELECT   FIRST_NAME, LAST_NAME
FROM   EMPLOYEES
WHERE  ( department_id, salary )  IN
                                  ( SELECT department_id,  MAX(salary)
                                    FROM EMPLOYEES
                                    GROUP BY department_id  )
```

Любой подзапрос, в свою очередь, также может обращаться к вложенному подзапросу. Например, получим список сотрудников, которых приняли на работу после того, как в штате компании появился менеджер отдела продаж (department_name='Sales'):

```
SELECT   FIRST_NAME, LAST_NAME
FROM   EMPLOYEES
WHERE   hire_date >
          ( SELECT   hire_date  FROM   EMPLOYEES
            WHERE   employee_id =
                        ( SELECT  manager_id  FROM DEPARTMENTS
                          WHERE  department_name='Sales' )  );
```

Следует отметить, что в рассмотренных примерах взаимодействие между вложенным подзапросом и запросом, который его охватывает, проходило по достаточно простой схеме – снизу вверх. Подзапрос такого типа называют **простым**, т.к. он самостоятельно выполняется всего один раз.

Наряду с этим, возможны и другие варианты. Например, получим список департаментов (с указанием department_id и department_name), которые размещаются на территории США (country_id ='US'):

```
SELECT  department_id,  department_name
FROM   DEPARTMENTS  dps
```

```
WHERE 'US' = ( SELECT country_id FROM LOCATIONS
                 WHERE location_id = dps.location_id )
```

Для этого примера важно отметить одну принципиальную особенность: внешний (основной) запрос управляет работой вложенного подзапроса, т.е. здесь происходит взаимодействие по схеме сверху вниз. В частности, при работе подзапроса нужно иметь конкретное значение **location_id**, которое передается из основного запроса. Следовательно, подзапрос выполняется несколько раз при разных значениях этого параметра. Подзапрос, обладающий такой особенностью, называют **связанным** (или *коррелированным*).

Логический оператор **EXISTS** позволяет проверить, дает ли подзапрос требуемый результат. Например, сформируем перечень департаментов, в которых отсутствуют данные о сотрудниках:

```
SELECT department_id, department_name
FROM DEPARTMENTS dps
WHERE not EXISTS
                ( SELECT * FROM EMPLOYEES
                  WHERE department_id = dps.department_id )
```

Следует обратить внимание, что здесь использован коррелированный подзапрос.

Варианты заданий

Задание 1. Получить список сотрудников, у которых период работы в компании ниже среднего стажа.

Задание 2. В списке сотрудников со стажем выше среднего указать, сколько лет не хватает им до достижения максимального стажа.

Задание 3. Получить список сотрудников для департаментов, у которых код местоположения (LOCATION_ID) отличается от 1500, 1700 и 2500.

Задание 4. Получить список, в который из каждого департамента выделить только сотрудников с максимальным стажем работы (**старожилы-могикане**).

Задание 5. Получить список департаментов, в которых средний стаж работы выше среднего по всей компании.

Задание 6. Получить список сотрудников департамента, для которого POSTAL_CODE=’26192’.

Задание 7. Получить список департаментов (с указанием department_id и department_name), которые размещаются на территории Европы (region_name ='Europe').

Задание 8. Сформировать перечень департаментов, в которых имеются сотрудники с именем John.

Лабораторное занятие № 11. «Создание запросов и процедур на изменение структуры базы данных»

Запрос **SELECT** будет реализован с помощью операции соединения таблиц (**JOIN**), если в разделе **FROM** через запятую перечислить несколько

источников данных. Такой синтаксис, относящийся к «старому стилю», определяет каждую операцию соединения двух таблиц в неявном виде.

Это связано с тем, что дополнительно требуется указать логическое выражение, которое должно содержаться в разделе **WHERE**. Именно здесь задаются **условия соединения отдельных строк** (записей) из исходных таблиц и, тем самым, конкретно определяется вид соединения.

Например, по таблицам EMPLOYEES и JOBS получим общий список сотрудников с указанием полного названия должности для каждого сотрудника.

```
SELECT  FIRST_NAME, LAST_NAME, JOB_TITLE
FROM  EMPLOYEES  E,  JOBS  J
WHERE  E.JOB_ID = J.JOB_ID
```

В этом запросе для сокращенного обозначения таблиц используются их псевдонимы – **E** и **J**.

Наряду с условием сцепления строк (записей) из исходных таблиц, раздел **WHERE** может содержать и дополнительные условия, по которым происходит отбор результатов соединения.

Соединения, при которых записи из исходных таблиц связываются по условию совпадения значений в заданных полях, относятся к классу **внутренних соединений**. Их называют также **эквисоединениями (equijoin)**, т.е. соединениями по условию равенства.

Чтобы получить **внешнее соединение,** нужно к имени поля, в котором ожидается отсутствие совпадающих значений, добавить **(+)**. Например, по таблицам DEPARTMENTS и EMPLOYEES требуется сформировать список сотрудников для каждого департамента, причем список должен включать даже те департаменты, в которых отсутствуют данные о сотрудниках:

```
SELECT  department_name d_name, last_name, first_name
FROM  departments D,  employees  E
WHERE  D.department_id = E.department_id (+)
ORDER BY d_name;
```

С помощью раздела **GROUP BY** можно произвести группировку результатов соединения таблиц.

В случае необходимости для таблицы можно произвести *самосоединение*, т.е. соединить таблицу саму с собой. Например, по таблице EMPLOYEES получим список сотрудников с указанием для каждого из них имени и фамилии его непосредственного начальника.

```
SELECT  S.last_name || ' ' || S.first_name  AS  Slave,
                B.last_name || ' ' || B.first_name  AS  Boss
FROM  employees  S,  employees  B
WHERE  S.manager_id = B.employee_id
ORDER BY  Boss;
```

Начиная с СУБД Oracle 9i и стандарта SQL:1992, стало возможным явным образом в разделе **FROM** оператора **SELECT** указывать операцию соединения в виде следующей конструкции:

⟨left_tab⟩ ⟨join_type⟩ ⟨right_tab⟩ **ON** ⟨join_cond⟩

Здесь подразумевается соединение таблиц ⟨**left_tab**⟩ и ⟨**right_tab**⟩, а элемент

⟨join_cond⟩ – это условие для соединения строк, которое переносится внутрь раздела **FROM.**

При таком синтаксисе в разделе **WHERE** остаются только дополнительные условия для отбора записей по другим критериям, что делает текст запроса более понятным для восприятия.

Синтаксический элемент **⟨join_type⟩** может принимать следующие значения:

- **[** INNER **]** JOIN — внутреннее соединение (применяется по умолчанию);
- LEFT **[** OUTER **]** JOIN — левое внешнее соединение;
- RIGHT **[** OUTER **]** JOIN — правое внешнее соединение;
- FULL **[** OUTER **]** JOIN — полное внешнее соединение.

Например, рассмотрим запрос для получения списка департаментов, которые размещаются на территории Канады (country_id ='CA').

```
SELECT   department_id, department_name
FROM   departments D  JOIN  locations L
                              ON   (D.location_id = L.location_id)
WHERE   country_id='CA';
```

В данном случае имеет место эквисоединение, причем совпадение значений проверяется в столбцах с одинаковыми названиями**.** По этой причине для раздела **FROM** есть более простая конструкция:

FROM departments **JOIN** locations **USING** (location_id)

Следует отметить, что во многих случаях операция **JOIN** исключает необходимость применения вложенных запросов.

В конструкции **ON ⟨join_cond⟩** можно указать соединение строк по ***произвольному*** условию с применением любых операций сравнения (<=, <>, LIKE, BETWEEN и др.). Это позволяет реализовать так называемое **Θ-*соединение***.

Например, получим распределение сотрудников по разным категориям в зависимости от уровня оклада (salary). Предполагается, что эти категории заданы в виде специальной таблицы **Sal_Grade**:

CAT_ID	LOW_LIMIT	HIGH_LIMIT
1	2000	5000
2	5001	10000
3	10001	15000
4	15001	20000
5	20001	25000

```
SELECT Low_limit, High_limit, count(*)  Freq
FROM  employees  JOIN  sal_grade
                          ON (salary BETWEEN Low_limit AND High_limit)
GROUP BY   Low_limit,  High_limit
ORDER BY   Low_limit;
```

Самый простой вариант построения раздела **FROM** – естественное соединение двух таблиц:

FROM ⟨left_tab⟩ **NATURAL JOIN** ⟨right_tab⟩

Такая конструкция очень привлекательна в силу своей простоты и подразумевает, что при соединении таблиц равенство значений будет проверяться попарно во всех столбцах с одинаковыми названиями. Однако нужно помнить об опасностях, которые возникают из-за **отсутствия явного контроля** за реальным условием соединения строк.

Конкретным примером такой опасности является следующий запрос для получения списка сотрудников по каждому департаменту:

SELECT department_name, last_name, first_name
FROM departments D **NATURAL JOIN** employees E
ORDER BY 1.

Здесь по правилам NATURAL JOIN формально (по умолчанию) применяется следующее условие соединения строк из рассматриваемых таблиц:

E.department_id = D.department_id AND E.manager_id = D.manager_id

Однако при построении требуемого списка нужна только левая часть этого условия, а правая часть вообще не отражает логику связей между таблицами departments и employees. В результате запрос дает список только тех сотрудников, у которых непосредственный начальник — менеджер подразделения.

Варианты заданий

Задание 1. По таблицам DEPARTMENTS и LOCATIONS получить список департаментов, которые размещены на территории США (country_id ='US').

Задание 2. Выдать список департаментов, указав по каждому департаменту число работающих сотрудников и общий фонд зарплаты. Список должен включать даже те департаменты, в которых данные о сотрудниках отсутствуют.

Задание 3. По таблице EMPLOYEES получить список сотрудников с указанием для каждого из них фамилии, имени и телефонного номера его непосредственного начальника. Включить в список даже тех сотрудников, у которых нет начальников в штате компании.

Задание 4. По таблицам DEPARTMENTS и EMPLOYEES с помощью внешнего соединения получить список сотрудников для каждого департамента. Выбрать наиболее подходящий тип соединения.

Задание 5. Получить список сотрудников, которые относятся к службам Marketing, Sales и IT.

Задание 6. Получить список сотрудников департамента, для которого POSTAL_CODE=’26192’.

Задание 7. Получить распределение сотрудников по разным категориям в зависимости от стажа работы. Эти категории задать в виде специальной таблицы **Stg_Grade**.

Задание 8. С помощью естественного соединения таблиц EMPLOYEES и JOBS найти среднюю заработную плату по отдельным должностям.

Команда INSERT. С помощью этой команды осуществляется добавление (вставка) новых записей в таблицу. При вставке единственной

строки рассматриваемая команда имеет следующий формат:

INSERT INTO ⟨ имя_таб ⟩ **[(**⟨ список_столбцов ⟩**)]**
VALUES (⟨ список_значений ⟩ **)**

В этом случае необходимо учитывать следующие **ограничения**:

- между позициями в списке столбцов и списке значений должно существовать строгое соответствие;
- в списке столбцов можно не указывать только те столбцы, для которых допускается значение **NULL** или существует значение по умолчанию (**DEFAULT**);
- соответствующие элементы в списке столбцов и списке значений должны быть совместимы по типу данных.

Для примера приведем команду вставки единственной строки в таблицу EMPLOYEES:

```
INSERT INTO EMPLOYEES ( EMPLOYEE_ID, LAST_NAME,
                        EMAIL, HIRE_DATE, JOB_ID, SALARY)
VALUES ( 300, 'Ivanov', 'IVAN', TO_DATE('1.06.2013'),
                        'IT_PROG', 3000);
```

Если при записи рассматриваемой команды список столбцов вообще не указан, то очевидно, что список значений должен обязательно содержать значения (VALUES) для всех столбцов (в порядке их описания на этапе создания таблицы). Например:

```
INSERT INTO EMPLOYEES
VALUES (302, 'John', 'Lemon', 'JLEMON', NULL,
              TO_DATE('15.06.2013'), 'IT_PROG', 3000, NULL, 107, 60)
```

Необходимо отметить, что такой вариант директивы **INSERT** считается **очень ненадежным** по следующим причинам:

- легко допустить ошибку, когда не видно списка названий столбцов таблицы;
- после изменения структуры таблицы операция становится некорректной.

Директива **INSERT** обеспечивает также вставку множества строк, которые копируются из другой таблицы. При этом добавляемые строки являются результатом работы подзапроса SELECT, входящего в состав директивы **INSERT**:

INSERT INTO ⟨ имя_таб ⟩ **[(**⟨ список_столбцов ⟩**)]**
SELECT . . . . . .

Вполне естественно, что при добавлении (вставке) множества_новых записей в некоторую таблицу также действуют все указанные выше ограничения.

Пусть, например, создана временная таблица EMP_TEMP со столбцами EMPLOYEE_ID, FIRST_NAME, LAST_NAME и SALARY. Тогда директива

```
INSERT INTO EMP_TEMP
    SELECT EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL
    FROM EMPLOYEES WHERE DEPARTMENT_ID IN(20, 50, 70);
```

позволяет скопировать в эту таблицу данные из таблицы EMPLOYEES о тех

сотрудниках, для которых значения DEPARTMENT_ID принадлежат списку (20, 50, 70).

Интересно, что с помощью единственного оператора **INSERT** 2-го можно распределить результаты запроса сразу по нескольким таблицам:

```
INSERT ALL
    WHEN  SALARY > 8000 THEN INTO EMP_HIGH
    WHEN  SALARY BETWEEN 4000 AND 8000
                                THEN INTO EMP_MID
    WHEN  SALARY < 4000 THEN INTO EMP_LOW
SELECT  *  FROM  EMPLOYEES
```

В этом случае после выполнения записанного оператора языка SQL в таблицы EMP_LOW и EMP_HIGH попадут, соответственно, данные о сотрудниках с низкой (SALARY<4000$) и высокой зарплате (SALARY>8000$), а таблица EMP_MID будет содержать данные о сотрудниках со средним уровнем зарплаты (8000$≥SALARY≥4000$). Предполагается, что все таблицы, которые участвуют в рассмотренном запросе, одинаковы по своей структуре.

Команда UPDATE. Эта команда позволяет изменить содержимое существующих записей в указанной таблице:

UPDATE ⟨ имя_таб ⟩
SET col_name1 = expr1 **[,** col_name2 = expr2 … **]**
[WHERE ⟨ условия_отбора_записей ⟩ **]**

Здесь необязательный раздел **WHERE** предназначен для того, чтобы изменить данные только в тех записях, которые удовлетворяют определенным условиям отбора.

Например, команда

```
UPDATE EMPLOYEES SET SALARY=SALARY*1.05
```

позволяет повысить оклад на 5% всему персоналу компании. В отличие от этого, команда

```
UPDATE EMPLOYEES SET SALARY=4000
WHERE EMPLOYEE_ID=300
```

установит новую зарплату в размере 4000$ только одному сотруднику, у которого EMPLOYEE_ID=300.

В составе оператора UPDATE могут присутствовать подзапросы, с помощью которых определяются подставляемые значения. Пусть, например, сотруднику, у которого EMPLOYEE_ID=300, требуется установить такие же значения DEPARTMENT_ID и PHONE_NUMBER, как в случае EMPLOYEE_ID=104. Эти изменения данных можно произвести с помощью следующего оператора:

```
UPDATE  EMPLOYEES  SET  (DEPARTMENT_ID, PHONE_NUMBER)=
        ( SELECT  DEPARTMENT_ID,  PHONE_NUMBER
          FROM  EMPLOYEES  WHERE EMPLOYEE_ID=104 )
WHERE  EMPLOYEE_ID=300
```

Команда DELETE. С помощью этой команды можно удалить записи из указанной таблицы:

DELETE FROM ⟨ имя_таб ⟩
[WHERE ⟨ условия_отбора_записей ⟩ **]**

В разделе **WHERE**, с помощью которого определяются условия для отбора удаляемых записей, могут присутствовать подзапросы.

Для полного удаления строк существует более быстрая команда:

TRUNCATE TABLE ⟨ имя_таб ⟩

Высокая скорость выполнения этой команды обусловлена следующими факторами:

- строки удаляются не поштучно, как при DELETE, а путем «усечения» сегмента (зоны) для хранения данных;
- информация, которая обеспечивает отмену этой операции, **не сохраняется**.

Варианты заданий

Задание 1. Добавить в таблицу JOBS новую запись, указав при этом следующие значения: JOB_ID=’HR_MAN’, JOB_TITLE=’Human Resources Manager’ и MIN_SALARY=4500.

Задание 2. Добавить новую запись в таблицу JOB_HISTORY, указав при этом EMPLOYEE_ID=111, START_DATE=TO_DATE (‘28.09.97’), END_DATE=TO_DATE(‘31.12.09’).

Задание 3. Запрос на добавление данных о новом сотруднике построить так, чтобы для создаваемой учетной записи сотрудника в столбец SALARY автоматически добавлялось минимальное значение заработной платы для указанной должности.

Задание 4. Запишите директиву языка SQL для добавления новой записи в таблицу COUNTRIES без указания списка столбцов этой таблицы.

Задание 5. Добавить в таблицу EMP_TEMP данные о сотрудниках, которые занимаются закупками товаров (Purchasing) и их продажей (Sales).

Задание 6. Используя в качестве источника записей соединение таблиц LOCATIONS и COUNTRIES, заполнить с помощью единственного оператора **INSERT** таблицы LOCS_1, LOCS_2 и LOCS_3, поместив туда данные, которые относятся к Европе (REGION_ID=1), Америке (REGION_ID=2) и Азии (REGION_ID=3), соответственно. Структуру таблиц LOCS_1, LOCS_2 и LOCS_3 выбрать по своему усмотрению.

При выполнении этого задания рекомендуется сначала воспользоваться следующим оператором для создания новой таблицы:

```
CREATE TABLE LOCS_1
  AS SELECT * FROM LOCATIONS NATURAL JOIN COUNTRIES
       WHERE 1=2
```

После выполнения этого оператора появится пустая таблица LOCS_1, структура которой будет объединять столбцы исходных таблиц LOCATIONS и COUNTRIES. Теперь остается вставить в нее требуемые записи. Одновременно заметим, что если в подзапросе, который применяется для создания новой таблицы, вместо символа * указать только некоторые

столбцы исходных таблиц, то новая таблица будет содержать только эти столбцы.

Задание 7. Повысить оклад на 10% всем руководителям департаментов.

Задание 8. Сотрудника, у которого LAST_NAME='Kozlov', перевести в департамент Accounting на должность Accounting Manager и установить ему зарплату в размере 8500$.

Задание 9. Сотрудника, у которого EMPLOYEE_ID=300, перевести в подчинение к тому же менеджеру, как у EMPLOYEE_ID=105, и установить зарплату на 10% выше минимума для занимаемой должности.

Задание 10. В таблице EMP_TEMP оставить только данные о сотрудниках, которые занимаются продажами (Sales) и маркетингом (Marketing).

Задание 11. В таблице EMP_LOW оставить только данные о сотрудниках, у которых заработная плата ниже половины средней зарплаты по компании.

4. ВОПРОСЫ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. Что обозначает аббревиатура SQL?

2. Будет ли правильным утверждать, что SQL является непроцедурным языком?

3. На какие категории разделяются команды языка SQL?

4. Какие функции выполняют команды управления транзакциями?

5. Перечислите основные предложения оператора SELECT. Какие из них являются обязательными?

6. Проанализируйте корректность следующих запросов:

   a) Select *
   b) Select * from checks
   c) Select amount name payee FROM checks.

7. Какая из агрегатных функций SUM, COUNT, MIN, MAX, AVG возвращает множество значений?

8. Будут ли корректными следующие утверждения:

   a) степень вложенности подзапросов не может превышать 2;
   b) коррелированные подзапросы являются полностью самостоятельными (независимыми).

9. Можно ли в выражении для ключевого слова WHERE задать несколько условий?

10. Будет ли правильным утверждение, что при использовании ключевого слова IN проверяемое значение должно совпадать с каждым элементом списка?

11. Будет ли правильным утверждение, что при наличии ключевого слова NAVING необходимо также использовать ключевые слова GROUP BY?

12. Какой тип соединения таблиц необходимо использовать в запросе, чтобы извлечь записи из некоторой таблицы независимо от наличия связанных записей в другой таблице?

13. В чем состоит особенность натурального (естественного) соединения двух таблиц?

14. Каким будет результат, если при выборке из двух таблиц не указать условие для связывания их записей?

15. Какие разделы могут использоваться при составлении команды SELECT для выбора данных из таблицы?

16. Какая директива языка SQL используется для создания таблиц?

17. Какие основные типы данных могут использоваться при создании таблиц?

18. Что такое псевдоним (алиас) и каким образом он определяется?

19. Какой формат имеет SQL-команда для уничтожения таблицы?

20. Поясните форматы SQL-команды для вставки новых записей в таблицу.

21. Каким образом можно указать конкретные колонки таблицы в запросе на выборку данных?

22. Каким образом оператор SELECT позволяет осуществить выбор строк таблицы, удовлетворяющих заданным условиям?

23. Какие специальные символы могут использоваться в функции LIKE для сравнения по образцу?

24. Какое ключевое слово задает режим запрета вывода строк-дубликатов?

25. Какими способами в операторе SELECT можно задать условия для соединения таблиц?

26. Какие объекты базы данных обеспечивают функционирование старых приложений без их модификации в случай изменения структуры БД?

27. Можно ли создать представление (view), которое будет работать одновременно с несколькими таблицами БД?

28. Каким образом можно перенести (по заданному условию) данные из одной таблицы в другую?

29. Каким целям служат представления? Чем они функционально отличаются от таблиц?

30. Как NULL-значения влияют на результаты арифметических операций?

31. Как правильно выполнять сравнение с NULL-значением?

32. Перечислите основные агрегатные функции и правила их использования.

33. Можно ли выполнить запрос на выборку, если данные расположены в нескольких таблицах?

34. В чём разница между INNER JOIN и OUTER JOIN?

35. Объясните отличие в использовании предложений WHERE и HAVING.

36. Укажите причины, по которым операция добавления в таблицу новой записи может завершиться с ошибкой.

37. Можно ли одной командой добавить в таблицу несколько новых строк?

38. Как быстро очистить всю таблицу? В чём недостаток этой операции?

39. Укажите реляционную операцию (UNION, UNION ALL, INTERSECT, MINUS) для работы с результатами двух запросов на выборку, если требуется:

a) показать совпадающие записи;
b) показать все данные;
c) показать все данные, но без повторений;
d) показать только те строки первого запроса, которые не выдаются вторым.

40. Поясните, в чем разница между командами DROP TABLE TAB_1 и DELETE FROM TAB_1.

5. СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

Основная учебная литература

1. Базы данных. Практическое применение СУБД SQL и NoSOL-типа для применения проектирования информационных систем: Учебное пособие / Мартишин С.А., Симонов В.Л., Храпченко М.В. – М.:ИД ФОРУМ, НИЦ ИНФРА-М, 2017. – 368 с. - Режим доступа: http://znanium.com/go.php?id=556449.
2. Базы данных: учебник / Л.И. Шустова, О.В. Тараканов. — М. : ИНФРА-М, 2018. — 304 с. + Доп. материалы. — (Среднее профессиональное образование). – Режим доступа: http://znanium.com/go.php?id=967755.
3. Методы, модели, средства хранения и обработки данных: учебник / Э.Г. Дадян, Ю.А. Зеленков. — М.: Вузовский учебник: ИНФРА-М, 2017. — 168 с. – Режим доступа: http://znanium.com/go.php?id=543943.

Дополнительная учебная литература

4. Голицына О.Л. Информационные системы и технологии: учебное пособие для вузов / О.Л. Голицына, Н.В. Максимов, И.И. Попов. – М.: Форум: Инфра-М, 2018. – 400 с.
5. Современные базы данных. Часть 2: практические задания: Учебно-методическое пособие / Дадян Э.Г. - М.:НИЦ ИНФРА-М, 2017. - 68 с.: 60x90 1/16 ISBN 978-5-16-106525-9 (online). - Режим доступа: http://znanium.com/go.php?id=959288.
6. Современные базы данных. Основы. Часть 1: Учебное пособие / Дадян Э.Г. - М.:НИЦ ИНФРА-М, 2017. - 88 с.: 60x90 1/16 ISBN 978-5-16-106526-6 (online). - Режим доступа: http://znanium.com/go.php?id=959289.

6. ПЕРЕЧЕНЬ ИНФОРМАЦИОННЫХ РЕСУРСОВ

- Интернет-университет информационных технологий: www.intuit.ru.
- Портал Центра Информационных Технологий: www.citforum.ru.
- Электронная библиотечная система издательства «ИНФРА-М»: www.znanium.com.

7. УЧЕБНО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ

№ п/п	Темы дисциплины	Перечень учебно-методических материалов
1	Основы построения баз данных	1, 2, 3
2	Технология работы с данными в СУБД MS Access	4, 5, 6
3	Публикация данных в Internet и Intranet	4, 5, 6
4	Использование Access совместно с другими приложениями MS Office	4, 5, 6

5	Язык SQL и СУБД типа OracleXE 10G	1, 5