Содержание

Предисловие.

Методическое пособие предназначено для организации лабораторных работ по курсу «Базы данных». Целью проведения лабораторных работ является практическое изучение языка SQL на примере СУБД Oracle 10/11g XE. Данное пособие во многом основано на учебных материалах, разработанных непосредственно компанией Oracle для обучения работе с языком SQL в рамках проекта Oracle Academy.

Для проведения лабораторных работ каждому студенту должен быть предоставлен индивидуальный доступ к собственному экземпляру учебной базы данных. Это могут быть локально установленные в компьютерном классе экземпляры Oracle 10/11g XE с выделенными под каждого студента схемами. Или же студент может получить бесплатный доступ к облачной платформе Oracle http://apex.oracle.com и создать там свой экземпляр учебной базы.

Каждая лабораторная работа состоит из двух частей. Первая часть работы – теоретическая. На ней преподаватель объясняет изучаемую тему, приводит примеры. Настоятельно рекомендуется все приведенные в теоретической части примеры исполнять в учебной базе прямо по ходу объяснения. Вторая часть лабораторной работы – практическая. На ней студенты в индивидуальном порядке выполняют самостоятельные задания, которые идут после каждой работы. По итогам каждой лабораторной работы студенты обязаны написать отчет, содержащий все сформулированные SQL-запросы. Работу разрешается выполнять как в компьютерном классе, так и самостоятельно, но с обязательным использованием СУБД Oracle. Во втором случае результат выполнения лабораторной работы выносится на защиту.

Помимо данного методического пособия, студентам рекомендуется пользоваться дополнительной литературой и online-ресурсами для более глубокого теоретического понимания технологий баз данных и развития практических навыков работы с языком SQL:

1. Кузнецов С.Д., Базы Данных. Серия Университетский учебник. М.: Изд. Академия, 2012
2. Мишра С., Бьюли А., Секреты Oracle SQL. М.: Изд. Символ-Плюс, 2006
3. Oracle Academy, http://academy.oracle.com
4. Oracle Documentation Library 11gR2, http://www.oracle.com/pls/db112

Лабораторная работа №1. Знакомство с СУБД Oracle 11g.

Цель работы: Познакомиться с основами реляционной модели данных и СУБД ORACLE 11g. Научиться подключаться к учебной БД при помощи SQL*Plus и SQL Developer, просматривать содержимое схемы и формулировать простейшие запросы к БД.

Теоретическая часть.

Oracle – реляционная СУБД. Реляционная модель предполагает хранение данных в виде двумерных таблиц: каждая строка содержит информацию о некотором экземпляре сущности, которую описывает таблица, представленная в виде набора столбцов. Например, если мы захотим хранить в базе данных список студентов группы, то это будет таблица СТУДЕНТЫ из, например, трех столбцов: номер студента, фамилия, имя и содержащая столько строк, сколько студентов учится в этой группе:

СТУДЕНТЫ		
Ном_студ	Фамилия	Имя
1	Иванов	Петр
2	Петров	Иван
3	Сидорова	Мария
4	Николаева	Анна

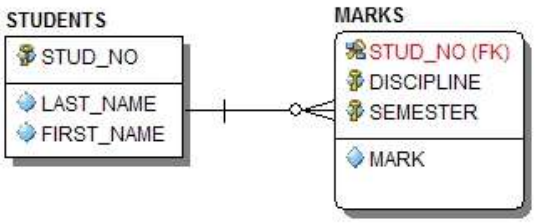
База данных, как правило, состоит из нескольких связанных между собой таблиц. Под связью понимается то, что некоторой строке из одной таблицы может соответствовать одна или несколько строк из другой таблицы.

СТУДЕНТЫ		
Ном_студ	Фамилия	Имя
1	Иванов	Петр
2	Петров	Иван
3	Сидорова	Мария
4	Николаева	Анна

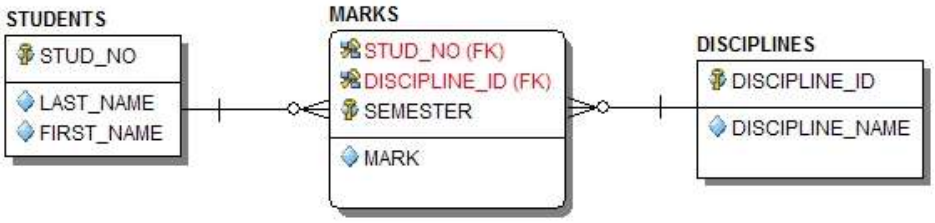
ОЦЕНКИ			
Ном_студ	Предмет	Семестр	Оценка
1	Математика	1	5
1	Физика	1	4
1	Английский	2	5
3	Математика	1	3
3	Физика	1	4
4	Математика	2	5

На приведенном примере видно, что таблицы СТУДЕНТЫ и ОЦЕНКИ связаны между собой по столбцу (ключу) Ном_студ. При этом для таблицы СТУДЕНТЫ этот столбец является уникальным (не может быть двух студентов с одинаковыми номерами) и называется первичным ключом (primary key). Для таблицы ОЦЕНКИ этот столбец не является уникальным (у одного студента может много оценок по разным предметам в разных семестрах), но может содержать только такие номера, которые есть в таблице СТУДЕНТЫ (иначе не выяснить, чьи это оценки) и называется внешним ключом (foreign key). Видно, что одному студенту может соответствовать как много оценок (например, у Иванова их 3),

так и только одна (Николаева) или вообще ни одной (Петров). Такая связь, допускающая соответствие одной строке основной таблицы нуля и более строк в подчиненной таблице, называется связь «один ко многим». Графически эти таблицы и эта связь отображаются на схеме БД следующим образом (имена таблиц и названия колонок в Oracle следует задавать латиницей):

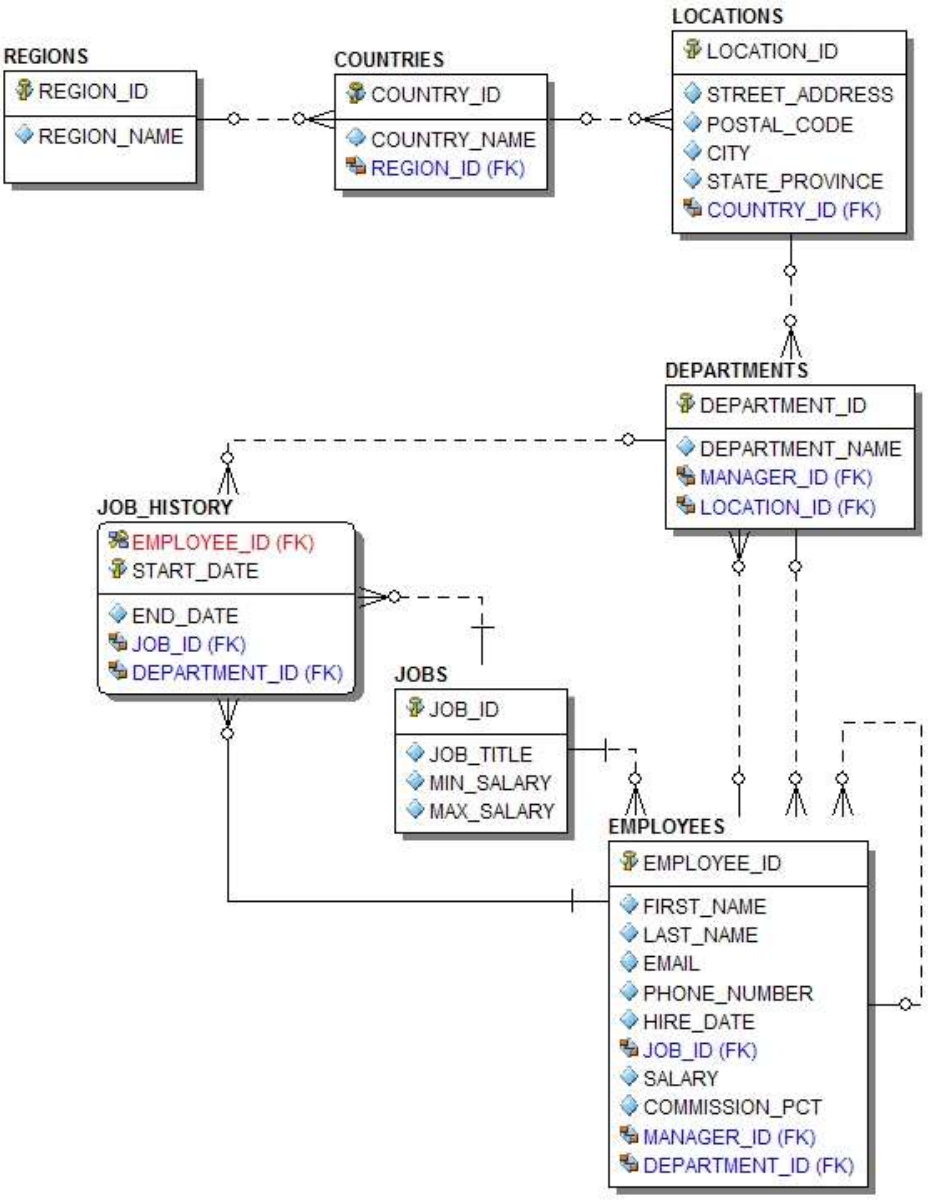


Легко заметить, что в таблице с оценками название предметов указаны словами, что при большом объеме таблице не очень удобно. Действительно, при заполнении тратим много времени и можем ошибиться в написании, при хранении – информация занимает много места, а при изменении, если изменилось название предмета, то надо будет обновить все строки с оценками. Поэтому в правильно спроектированной (нормализованной) базе данных для этого случая, скорее всего, будет предусмотрена отдельная таблица-справочник для предметов, а схема БД будет выглядеть следующим образом:



Для лабораторных работ будет использоваться специально разработанная Oracle учебная база данных, схема которой приведена ниже. Эта база данных содержит информацию о сотрудниках некоторой компании, имеющей сеть распределенных по миру офисов. Сотрудники работают в разных отделах компании на разных должностях и с разным уровнем оплаты, кроме этого хранится история смены работы сотрудниками. Прежде чем приступать к выполнению лабораторных работ следует внимательно изучить эту схему, чтобы понять, какая информация в каких таблицах хранится и как эти таблицы связаны между собой. Особое внимание обратите на связи таблицы EMPLOYEES, которые показывают, что столбец, являющийся внешним ключом, не обязательно называется так же, как столбец в основной таблице.

Схема учебной БД:



СУБД Oracle хранит данные и метаданные на внешних носителях специальным образом и не предоставляет прямого доступа к ним. Обратиться к данным можно только через сервер СУБД. Для этого необходимо установить соединение с сервером, после чего можно формулировать запросы к базе данных при помощи языка SQL. Для установки соединения необходимо клиентское программное обеспечение. Это может быть SQL Plus (терминальный клиент) или, например, SQL Developer – графическая оболочка, которая позволяет как работать с «чистым» SQL, так и более удобно просматривать и изменять содержимое схемы БД, а также работать с языком программирования PL/SQL. Если для обучения будет использоваться облачная платформа Oracle, то клиентом будет обычный Web Browser. Конкретный способ подключения к БД следует уточнить у преподавателя.

После подключения к учебной БД, попробуйте выполнить несколько простых команд.

```
DESC EMPLOYEES
```

Эта команда позволяет получить описание структуры таблицы сотрудников - EMPLOYEES. Помимо названия столбцов, эта команда показывает типы данных этих столбцов и возможность столбца содержать неопределенные значения.

```
SELECT * FROM CAT;
```

Эта команда позволяет получить список всех таблиц и других объектов базы данных. Обратите внимание, что SQL-команда сначала вводится (при это она может быть записана на нескольких строках), после чего завершается символом «;» и, для ее выполнения, необходимо или просто нажать Enter в терминальном режиме или же выделить написанное и нажать иконку исполнения запроса (RUN) в случае с SQL Developer или облачной платформой.

```
SELECT * FROM countries;
```

Эта команда позволит посмотреть все данные из таблицы COUNTRIES.

Для отсоединения от сервера можно просто закрыть клиентское приложение или использовать команду DISCONNECT.

Задания для самостоятельной работы.

1. Присоединитесь к учебной схеме БД.
2. Выведите структуру таблицы EMPLOYEES.
3. Выведите список всех таблиц в учебной БД.
4. Сколько полей символьного типа в таблице с историей смены работы?
5. Перечислите все внешние ключи таблицы DEPARTMENTS.

Лабораторная работа №2. Использование оператора SELECT.

Цель работы: Научиться использовать с базовый вариант оператора SELECT с выбором всех столбцов, выбором конкретных столбцов, использованием заголовков столбцов и арифметических выражений.

Теоретическая часть.

Базовый синтаксис оператора SELECT состоит из двух классов: SELECT и FROM. Класс SELECT определяет, что (какие столбцы) извлекать, а класс FROM указывает откуда (из какой таблицы) извлекать данные:

```
SELECT *|{[DISTINCT] столбец|выражение [псевдоним],...}
FROM таблица;
```

Для выбора всех столбцов используется символ "*", например:

```
SELECT * FROM departments;
```

Для выбора конкретных столбцов надо их перечислить в классе SELECT, например:

```
SELECT last_name, phone_number FROM employees;
```

Помимо имен столбцов, в SELECT можно использовать арифметические выражения, например:

```
SELECT last_name, salary, salary * 0.87 FROM employees;
```

По умолчанию заголовком столбца полученной выборки является имя столбца, но можно задать и другое имя для столбца выборки при помощи использования алиаса, например:

```
SELECT last_name, salary,
       salary * 0.87 as "Salary after tax"
FROM employees;
```

Ключевое слово "as" при задании алиаса можно опустить, а если алиас состоит только из одного слова, то можно не заключать его в кавычки:

```
SELECT last_name, salary, salary * 0.13 Tax FROM employees;
```

Помимо арифметических выражений в SELECT можно использовать оператор конкатенации строк (обозначается "||"). Соединять можно как содержимое столбцов, так и литеральные константы, которые следует заключать в апострофы, например:

```
SELECT first_name||' '||last_name as "Full Name"
FROM employees;
```

По умолчанию SELECT выдает все строки, включая возможные дубликаты:

```
SELECT first_name FROM employees;
```

Для устранения дубликатов можно использовать ключевое слово DISTINCT:

```
SELECT DISTINCT first_name FROM employees;
```

Задания для самостоятельной работы.

1. Выберите все данные из таблицы DEPARTMENTS.
2. Покажите структуру таблицы EMPLOYEES . Составьте запрос для вывода имени каждого служащего, должности, даты найма и номера.
3. Составьте запрос для вывода неповторяющихся должностей из таблицы EMPLOYEES.
4. Выполните запрос из задания 2, назвав столбцы Emp #, Employee, Job и Hire Date.
5. Выведите на экран имя, соединенное с идентификатором должности через запятую и пробел. Назовите новый столбец Employee and Title.

Лабораторная работа №3. Разделы WHERE и ORDER BY оператора SELECT.

Цель работы: Научиться использовать класс WHERE для ограничения выборки по строкам. Изучить операторы сравнения и логические операторы в WHERE, операторы LIKE, IN, BETWEEN. Научиться сортировке строк выборки при помощи ORDER BY.

Теоретическая часть.

Количество возвращаемых оператором SELECT строк можно ограничить при помощи класса WHERE, наложив условие, какие именно строки следует возвращать:

```
SELECT *|{[DISTINCT] столбец|выражение [псевдоним],...}
FROM таблица
[WHERE условие(я)];
```

Например, для того, чтобы выбрать всех сотрудников отдела 90:

```
SELECT * FROM employees WHERE department_id = 90;
```

При формулировании условий символьные заключаются в апострофы:

```
SELECT last_name, job_id, department_id
FROM employees
WHERE last_name = 'King';
```

Можно использовать различные условия сравнения:

```
SELECT last_name, salary
FROM employees
WHERE salary <= 3000;
```

Помимо простых условий сравнения, можно использовать более сложные конструкции:

Оператор	Значение
BETWEEN {A} AND {B}	Находится в диапазоне от A до B (включительно)
IN ({список})	Совпадает с каким-либо значением списка
LIKE {шаблон}	Подходит по символьный шаблон
IS NULL	Является неопределенным значением

```
SELECT last_name, salary
FROM employees
WHERE salary BETWEEN 2500 AND 3500;

SELECT employee_id, last_name, salary, manager_id
FROM employees
WHERE manager_id IN (100, 101, 201);
```

При формулировании шаблона для оператора LIKE, "%" – означает ноль или больше любых символов, "_" – ровно один любой символ. Нарпимер:

```
SELECT first_name
FROM employees
WHERE first_name LIKE 'S%';

SELECT last_name
FROM employees
WHERE last_name LIKE '_o%';
```

Для проверки на неопределенное значение используется специальный оператор IS NULL, т.к. оператор сравнения с неопределенностью (= NULL) всегда будет возвращать ложь:

```
SELECT last_name, manager_id
FROM employees
WHERE manager_id IS NULL;
```

Несколько простых условий внутри класса WHERE можно комбинировать в одно сложное при помощи логических операторов AND, OR и NOT:

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary >=10000
AND job_id LIKE '%MAN%';

SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary >= 10000
OR job_id LIKE '%MAN%';

SELECT last_name, job_id
FROM employees
WHERE job_id NOT IN ('IT_PROG', 'ST_CLERK', 'SA_REP');
```

Общий приоритет исполнения операторов выглядит следующим образом:

Приоритет	Оператор
1	Арифметические операторы
2	Конкатенация
3	Сравнение
4	IS [NOT] NULL, [NOT] LIKE, [NOT] IN
5	[NOT] BETWEEN
6	NOT
7	AND
8	OR

Таким образом, сначала выполняются арифметические операции, потом конкатенация и так далее, согласно приоритету, вплоть до логического оператора OR, который выполняется в последнюю очередь:

```
SELECT last_name, job_id, salary
FROM employees
WHERE job_id = 'SA_REP'
OR job_id = 'AD_PRES'
AND salary > 15000;
```

Изменить штатный приоритет исполнения операторов можно при помощи круглых скобок:

```
SELECT last_name, job_id, salary
FROM employees
WHERE (job_id = 'SA_REP'
OR job_id = 'AD_PRES')
AND salary > 15000;
```

По умолчанию оператор SELECT возвращает строки в произвольном (удобном СУБД) порядке. Для упорядочивания выборки используется класс ORDER BY, указываемый последним. Сортировать можно как возрастанию (ASC), так и по убыванию значений (DESC). Если порядок в явном виде не указан, применяется сортировка по возрастанию:

```
SELECT *|{[DISTINCT] столбец|выражение [псевдоним],...}
FROM таблица
[WHERE условие(я)]
[ORDER BY {столбец, выражение, псевдоним} [ASC|DESC]];
```

Например:

```
SELECT last_name, job_id, department_id, hire_date
FROM employees
ORDER BY hire_date;

SELECT last_name, job_id, department_id, hire_date
FROM employees
ORDER BY hire_date DESC;
```

Сортировать можно не только по названию столбца, но и по его алиасу или порядковому номеру в запросе:

```
SELECT employee_id, last_name, salary*12 annsal
FROM employees
ORDER BY annsal DESC;

SELECT employee_id, last_name
```

```
FROM employees
ORDER BY 2;
```

Сортировать можно по нескольким столбцам, указав их через запятую:

```
SELECT last_name, department_id, salary
FROM employees
ORDER BY department_id, salary DESC;
```

Сортировать можно даже и по столбцам, которые не вошли в SELECT-класс:

```
SELECT last_name, department_id
FROM employees
ORDER BY salary DESC;
```

Задания для самостоятельной работы.

1. Создайте запрос для вывода имени и заработной платы служащих, зарабатывающих более $12000.
2. Создайте запрос для вывода фамилии и номера отдела служащего под номером 176.
3. Выведите фамилии и оклады всех служащих, чей оклад не входит в диапазон от $5000 до $12000.
4. Выведите фамилию, идентификатор должности и дату начала работы всех служащих, нанятых в период с 20 февраля 1998 г. по 1 мая 1998 г. Отсортируйте данные в порядке возрастания даты найма. Для сравнения с датой используйте конструкцию вида TO_DATE('20.02.1998','dd.mm.yyyy').
5. Выведите фамилию и номер отдела всех служащих из отделов 20 и 50. Отсортируйте данные по фамилиям в алфавитном порядке.
6. Создайте запрос для вывода фамилий и окладов служащих отделов 20 и 50, зарабатывающих от $5000 до $12,000. Назовите столбцы Employee и Monthly Salary соответственно.
7. Выведите фамилии и должности всех служащих, не имеющих менеджера.
8. Выведите фамилию, оклад и комиссионные всех служащих, зарабатывающих комиссионные. Отсортируйте данные в порядке убывания окладов и комиссионных.
9. Выведите все фамилии служащих, в которых третья буква – "a".
10. Запросите фамилии, должности и оклады всех служащих, работающих торговыми представителя (SA_REP) или клерками на складе (ST_CLERK) и с окладом не $2500, $3500 или $7000.

Лабораторная работа №4. Использование однострочных функций.

Цель работы: Научиться использовать арифметические, строковые и другие функции в запросах. Понять принципы преобразования типов.

Теоретическая часть.

В SQL имеется два типа функций: однострочные и многострочные. Оба типа функций возвращают одно значение, но однострочные возвращают по одному значению на каждую строку, а многострочные возвращают одно значение на группу строк. Сейчас рассмотрим только однострочные функции.

Однострочные функции подразделяются на:

- Символьные, предназначенные для работы со строками символов, подразделяются на функции:
    - Преобразования регистра:
        - LOWER
        - UPPER
        - INITCAP
    - Манипулирования символами:
        - CONCAT
        - SUBSTR
        - LENGTH
        - INSTR
        - LPAD | RPAD
        - TRIM
        - REPLACE
- Числовые, предназначенные для работы с числами:
    - ROUND
    - TRUNC
    - MOD
- Общие, предназначенные для работы с любыми типами данных и неопределенными значениями.
    - Будут рассмотрены позднее.
- Преобразования, предназначенные для явного преобразования типов.
    - Будут рассмотрены позднее.
- Даты и времени, предназначенные для работы с типами данных даты/времени.
    - Будут рассмотрены позднее.

Подробное описание всех функций и их аргументов можно найти в справочной литературе по SQL, например [4], сейчас же будут рассмотрены основные функции и способы их применения.

Аргументами однострочной функции могут быть: константы, переменные, имена столбцов, выражения.

Рассмотрим пример использования функции преобразования регистра в классе WHERE. Данный запрос не дает результата:

```
SELECT employee_id, last_name, department_id
FROM employees
WHERE last_name = 'higgins';
```

Возвращается пустое множество строк потому, что в базе данных фамилии начинаются с заглавной буквы, хотя, в принципе, могут быть там сохранены в каком угодно виде. Для того, чтобы не зависеть от конкретного написания фамилии в базе данных, можно использовать функцию LOWER:

```
SELECT employee_id, last_name, department_id
FROM employees
WHERE LOWER(last_name) = 'higgins';
```

В приведенном примере можно было использовать и функции UPPER или INITCAP:

Функция	Результат
LOWER('SQL Course')	sql course
UPPER('SQL Course')	SQL COURSE
INITCAP('SQL Course')	Sql Course

Функции манипулирования символами:

Функция	Результат
CONCAT('Hello', 'World')	HelloWorld
SUBSTR('HelloWorld',1,5)	Hello
LENGTH('HelloWorld')	10
INSTR('HelloWorld', 'W')	6
LPAD(salary,10,'*')	*****24000
RPAD(salary, 10, '*')	24000*****
TRIM(' HelloWorld ')	HelloWorld
TRIM('H' FROM 'HelloWorld')	elloWorld

Функции можно использовать как в классе SELECT, так и в классе WHERE:

```
SELECT employee_id, CONCAT(first_name, last_name) NAME,
job_id,
LENGTH (last_name), INSTR(last_name, 'a') "Contains 'a'?"
FROM employees
WHERE SUBSTR(job_id, 4) = 'REP';
```

Числовые функции:

Функция	Результат
ROUND(45.926, 2)	45.93 (округление)

TRUNC(45.926, 2)	45.92 (усечение)
MOD(16, 3)	1 (остаток от деления)

Пример использования функции ROUND:

```
SELECT last_name, salary, ROUND(salary * 0.87, 2) "Salary
After Tax"
FROM employees
WHERE job_id = 'SA_REP';
```

Пример использования функции TRUNC (dual – фиктивная таблица, содержащая одну строку, которую удобно использовать для вычисления функций):

```
SELECT TRUNC(45.923,2), TRUNC(45.923), TRUNC(45.923,-1)
FROM DUAL;
```

Пример использования функции MOD (фамилии всех сотрудников с четными номерами):

```
SELECT last_name FROM employees
WHERE mod(employee_id,2)=0;
```

Обратите внимание, что в простых случаях, неявное преобразование символьных и числовых типов осуществляется автоматически, например, допустимы следующие конструкции:

```
SELECT last_name||' '||salary FROM employees;

SELECT last_name, salary * '0.87' FROM employees;
```

Преобразование типов можно задать и явно при помощи функций TO_CHAR, TO_NUMBER, TO_DATE. Например:

```
SELECT TO_NUMBER('1234.64', '9999.99') FROM dual;

SELECT last_name,salary FROM employees
ORDER BY TO_CHAR(salary);

SELECT last_name FROM employees
WHERE hire_date >= TO_DATE('01.01.2000','dd.mm.yyyy');
```

Задания для самостоятельной работы.

1. Выведите номер служащего, его фамилию, оклад и новый оклад, повышенный на 15% и округленный до целого. Назовите столбец New Salary.

2. Добавьте еще один столбец к запросу из задания 1, который будет содержать результат вычитания старого оклада из нового. Назовите столбец Increase.
3. Выведите фамилии служащих (первая буква каждой фамилии должна быть заглавной, а остальные – строчными) и длину каждой фамилии для тех служащих, фамилия которых начинается с символа J, A или M. Присвойте соответствующие заголовки столбцам.
4. Получите по каждому служащему отчет в следующем виде: <фамилия> зарабатывает <оклад> в месяц, но желает <утроенный оклад>. Назовите столбец Dream Salaries.

Лабораторная работа №5. Использование условных выражений.

Цель работы: Научиться работать со значением NULL с использованием выражений NVL, COALESCE. Научиться использовать выражения DECODE и CASE.

Теоретическая часть.

Для всех типов данных Oracle предусмотрено специальное значение – NULL. Это неопределенное значение - значение, которое недоступно, не присвоено, неизвестно или неприменимо. Это не ноль и не пробел.

Напомним, что для проверки на неопределенное значение используется специальный оператор IS [NOT] NULL, т.к. операторы сравнения с неопределенностью (= или <> NULL) всегда будут возвращать ложь:

```
SELECT last_name, manager_id
FROM employees
WHERE commission_pct = NULL;

SELECT last_name, manager_id
FROM employees
WHERE commission_pct IS NULL;
```

Результат вычисления выражения, содержащего неопределенное значение, также будет неопределенным, например:

```
SELECT last_name, commission_pct,
       salary*(1+commission_pct) full_sal
FROM employees;
```

Для корректной работы с неопределенными значениями применяются операторы IS NULL и IS NOT NULL, а также общие функции (функции, работающие со всеми типами данных):

Функция	Возвращает
NVL(arg1,agr2)	IF agr1 IS NOT NULL THEN agr1 ELSE agr2
COALESCE(agr1, agr2,..,agrN)	IF agr1 IS NOT NULL THEN agr1 ELSE IF agr2 IS NOT NULL THEN agr2 … ELSE agrN
NULLIF(agr1,arg2)	IF arg1 = agr2 THEN NULL ELSE agr1

Использование данных функций позволяет избавиться от неопределенностей при вычислении выражений:

```
SELECT last_name,
       salary + salary*nvl(commission_pct,0) full_sal
FROM employees;
```

Преимущество функции COALESCE по сравнению с функцией NVL состоит в том, что функция COALESCE может обрабатывать несколько альтернативных значений:

```
SELECT last_name,
       COALESCE(salary*(1+commission_pct),salary,0) full_sal
FROM employees;
```

Условные выражения CASE и DECODE позволяют применять логические конструкции IF-THEN-ELSE внутри команды SQL.

```
CASE выражение
  WHEN сравн_выражение1 THEN возвр_выражение1
 [WHEN сравн_выражение2 THEN возвр_выражение2
  WHEN сравн_выражениеn THEN возвр_выражениеn
  ELSE else_выражение]
END
```

Например:

```
SELECT last_name, job_id, salary,
  CASE job_id WHEN 'IT_PROG' THEN 1.10*salary
    WHEN 'ST_CLERK' THEN 1.15*salary
    WHEN 'SA_REP' THEN 1.20*salary
    ELSE salary
  END "REVISED_SALARY"
FROM employees;
```

```
DECODE(столбец|выражение, вариант1, результат1
                        [, вариант2, результат2,...,]
                                [, результат_по_умолчанию])
```

Например:

```
SELECT last_name, job_id, salary,
       DECODE(job_id,
              'IT_PROG',  1.10*salary,
              'ST_CLERK', 1.15*salary,
              'SA_REP',   1.20*salary,
                          salary) REVISED_SALARY
FROM employees;
```

Выражение DECODE удобно использовать для реализации небольших справочников прямо внутри SQL запроса, например, для вычисления прогрессивной ставки налога:

```
SELECT last_name, salary,
       DECODE (TRUNC(salary/2000, 0),
                 0, 0.00,
                 1, 0.09,
                 2, 0.20,
                 3, 0.30,
                 4, 0.40,
                 5, 0.42,
                 6, 0.44,
                 0.45) TAX_RATE
FROM employees
WHERE department_id = 80;
```

Задания для самостоятельной работы.

1. Выведите список сотрудников, у которых нет начальника.
2. Напишите запрос для вывода фамилии и суммы комиссионных каждого служащего. Если служащий не зарабатывает комиссионных, укажите в столбце "No Commission." Назовите столбец COMM.
3. Используя функцию DECODE, напишите запрос для отображения должности сотрудника и ее условного кода. Код для администрации (JOB_ID like 'AD%') равен "A", для IT (JOB_ID like 'IT%') равен "B", для маркетинга (JOB_ID like 'MK%') равен "C", а для всех остальных равен "D".
4. Перепишите команду из предыдущего задания, используя синтаксис выражения CASE.

Лабораторная работа №6. Работа с функциями даты и времени.

Цель работы: Научиться извлечению столбцов даты и времени в различных форматах представления. Изучить функции преобразования из/к дате и времени и арифметику даты и времени.

Теоретическая часть.

Oracle хранит данные во внутреннем цифровом формате: Век, год, месяц, число, часы, минуты, секунды. При запросе полей типа дата они возвращаются в определенном формате. Формат может быть задан явно, либо используется формат по умолчанию: DD-MON-RR (число-месяц-год).

```
SELECT last_name, hire_date
FROM employees
WHERE last_name like 'G%';
```

Определены две специальные функции, возвращающие текущую дату и время:

- SYSDATE – возвращает текущую дату и время для часового пояса базы данных.
- CURRENT_DATE – возвращает текущую дату и время для часового пояса пользователя, присоединившегося к базе данных.

```
SELECT to_char(current_date,'dd.mm.yyyy hh24:mi:ss')
       usr_date,
       to_char(sysdate,'dd.mm.yyyy hh24:mi:ss')
       sys_date FROM dual;

-- Изменим временную зону пользователя и повторим запрос
ALTER session SET time_zone = '+01:00';

SELECT to_char(current_date,'dd.mm.yyyy hh24:mi:ss')
       usr_date,
       to_char(sysdate,'dd.mm.yyyy hh24:mi:ss')
       sys_date FROM dual;

-- вернем значение по умолчанию
ALTER session SET time_zone = dbtimezone;
```

Даты можно сравнивать между собой, а также с датами можно осуществлять арифметические операции:

- Результатом прибавления числа к дате и вычитания числа из даты является дата.
- Результатом вычитания одной даты из другой является количество дней, разделяющих эти даты.
- Прибавление часов к дате производится путем деления количества часов на 24.

```
SELECT last_name, (SYSDATE-hire_date)/7 AS WEEKS
FROM employees
WHERE department_id = 90;

SELECT to_char(CURRENT_DATE-3/24,'dd.mm.yyyy hh24:mi:ss')
FROM DUAL;
```

Для работы с датами предусмотрены специальные функции:

Функция	Описание
MONTHS_BETWEEN	Число месяцев, разделяющих две даты.
ADD_MONTHS	Добавление календарных месяцев к дате.
NEXT_DAY	Ближайшая дата, когда наступит заданный день недели.
LAST_DAY	Последняя дата текущего месяца.
ROUND	Округление даты.
TRUNC	Усечение даты.

Например:

```
SELECT MONTHS_BETWEEN('01-SEP-05','11-JAN-04') FROM dual;

SELECT ADD_MONTHS ('11-JAN-04',6) FROM dual;

SELECT NEXT_DAY ('02-SEP-13','FRIDAY') FROM dual;

SELECT LAST_DAY('01-FEB-05') FROM dual;

SELECT ROUND(SYSDATE,'MONTH') FROM dual;

SELECT ROUND(SYSDATE ,'YEAR') FROM dual;

SELECT TRUNC(SYSDATE ,'YEAR') FROM dual;

SELECT TO_CHAR(sysdate,'dd.mm.yyyy hh24:mi:ss'),
       TO_CHAR(TRUNC(sysdate), 'dd.mm.yyyy hh24:mi:ss')
FROM dual;
```

Для задания даты и опередение формата вывода следует использовать функции TO_CHAR и TO_DATE, не надо полагаться на формат даты по умолчанию, т.к. он может быть изменен:

```
SELECT hire_date FROM EMPLOYEES
WHERE hire_date > '1-JAN-99';

-- Меняем формат даты по умолчанию и убеждаемся,
-- что формат вывода изменился и
```

```
-- условие выборки стало некорректным
alter session set NLS_DATE_FORMAT = 'DD.MM.YYYY HH24:MI:SS';

SELECT hire_date FROM EMPLOYEES
WHERE hire_date > '1-JAN-99';

-- Возвращаем формат даты
alter session set NLS_DATE_FORMAT = 'DD-MON-RR';
```

Если требуется задать в запросе конкретную дату, используйте функцию TO_DATE:

```
SELECT last_name, hire_date FROm employees
WHERE hire_date >= to_date('01.01.1999','dd.mm.yyyy');
```

Если требуется вывести дату/время в нужном формате – используйте функцию TO_CHAR:

```
SELECT last_name,to_char(hire_date,'dd.mm.rr / day') FROM
employees;
```

Допустимые элементы формата даты:

- YYYY - Полный год цифрами
- YEAR - Год прописью
- MM - Двузначное цифровое обозначение месяца
- MONTH - Полное название месяца
- DY - Трехзначное алфавитное сокращенное название дня недели
- DAY - Полное название дня недели
- DD - Номер дня месяца
- HH – Часы в формате 12 am/pm
- AM – До/после полудня
- HH24 – Часы в формате 24
- MI – Минуты
- SS - Секунды

Например:

```
SELECT to_char(sysdate,'dd "of" month yyyy, hh:mi am') FROM
dual;
```

Для работы с представлением года двумя цифрами предусмотрен специальный формат RR, который, в отличие от YY будет возвращать разное значение, в зависимости от текущего года. Например, если сейчас 2013й год, то to_date('01.01.95','dd.mm.yy') = 1 января 2095 года, а то to_date('01.01.95','dd.mm.rr') = 1 января 1995 года. А если бы на дворе был, например, 2055 год, то эти два вызова дали бы одинаковый результат. Таким образом, формат RR позволяет трактовать двузначное представление года как наиболее

разумное в текущий момент: в начале века (до 50го года) год, меньший 50 считается текущим, а больший – предыдущим веком, а в конце – наоборот. Формат же YY всегда подразумевает текущий век:

```
SELECT to_char(to_date('01.09.13','dd.mm.rr'),'dd.mm.yyyy')
       "13 RR",
       to_char(to_date('01.09.13','dd.mm.yy'),'dd.mm.yyyy')
       "13 YY",
       to_char(to_date('01.09.95','dd.mm.rr'),'dd.mm.yyyy')
       "95 RR",
       to_char(to_date('01.09.95','dd.mm.yy'),'dd.mm.yyyy')
       "95 YY"
FROM dual;
```

Помимо типа данных DATE, позволяющего хранить дату и время с точностью до секунды, в Oracle предусмотрены типы данных для более точного хранения даты и времени:

- TIMESTAMP
- TIMESTAMP WITH TIME ZONE
- TIMESTAMP WITH LOCAL TIME ZONE

И функции для работы с ними: CURRENT_TIMESTAMP, LOCALTIMESTAMP, TO_TIMESTAMP. Более подробную информацию по этим типам данных можно найти в документации Oracle.

Задания для самостоятельной работы.

1. Напишите запрос для вывода текущей даты. Назовите столбец Date.
2. Для каждого служащего выведите фамилию и вычислите количество месяцев со дня найма до настоящего времени, округленное до ближайшего целого. Назовите столбец MONTHS_WORKED. Результаты отсортируйте по количеству отработанных месяцев. Округлите число месяцев до ближайшего целого.
3. Для каждого служащего выведите фамилию, дату найма и дату пересмотра зарплаты, которая приходится на первый понедельник после шести месяцев работы. Назовите столбец REVIEW.
4. По каждому служащему выведите фамилию, дату найма и день недели, когда он был нанят на работу. Назовите последний столбец DAY. Отсортируйте результаты по дням недели, начиная с понедельника.
5. Установите в сеансе параметр NLS_DATE_FORMAT в DD-MON-YYYY HH24:MI:SS.
6. Измените в сеансе значение параметра TIME_ZONE на '-07:00'.
7. Выведите в сеансе значения функций CURRENT_DATE, CURRENT_TIMESTAMP и LOCALTIMESTAMP.

Лабораторная работа №7. Запросы из нескольких таблиц.

Цель работы: Научиться формулировать запросы более чем к одной таблице, использовать алиасы таблиц. Узнать, что такое декартово произведение и изучить различные варианты JOIN.

Теоретическая часть.

Команда SELECT позволяет осуществлять выборку не только из одной, но и из 2х и более таблиц. Все таблицы, из которых осуществляется выборка перечисляются в классе FROM, а условия связи формулируются в классе WHERE.

```
SELECT таблица1.столбец, таблица2.столбец
FROM таблица1, таблица2
WHERE таблица1.столбец1 = таблица2.столбец2;
```

Например, для того, чтобы получить список сотрудников с указанием названий департаментов, в которых они работают, надо соединить две таблицы: EMPLOYEES и DEPARTMENTS:

```
SELECT employees.last_name, departments.department_name
FROM employees, departments
WHERE employees.department_id = departments.department_id;
```

Если условие соединения опущено, то запрос из двух таблиц даст декартово произведение этих таблиц (картезианскую выборку), т.е. каждой строке из первой таблицы будут сопоставлены все строки из второй таблицы. И, если первая таблица содержит 10 строк, а вторая – 20, то в результате запроса получится 200 строк. Поэтому, во избежание получения декартова произведения (что на практике требуется очень редко), предложение WHERE всегда должно включать правильное условие соединения.

Выделяют несколько типов соединений таблиц. Приведенный выше пример соединения – это эквисоедиение, которое применяется при соединении таблиц по внешнему ключу. Действительно, department_id – это внешний ключ для таблицы employees, что гарантирует, что одной строке в таблице employees будет соответствовать не более одной строки в таблице departments (если employees.department_id is null, то, очевидно, никакой строки неопределенному значению не соответствует).

Эквисоединение может быть дополнено обычным условием, например, если нам требуется распечатать список сотрудников, принятых на работу после 1 января 2000 года:

```
SELECT employees.last_name, departments.department_name
FROM employees, departments
WHERE employees.department_id = departments.department_id
AND hire_date > TO_DATE ('01.01.00','dd.mm.rr');
```

Для различения одноименных столбцов из разных таблиц используются префиксы в виде имен таблиц. Одноименные столбцы из разных таблиц можно различать по их псевдонимам. Использование псевдонимов упрощает запросы и увеличивает производительность, т.к. компилятору не приходится разбирать, какой из столбцов имеется в виду:

```
SELECT e.employee_id, e.last_name, e.department_id,
d.department_id, d.location_id
FROM employees e, departments d
WHERE e.department_id = d.department_id;
```

При соединении 2х таблиц, требует как минимум одно условие, при соединении N таблиц, требуется N-1 условие:

```
SELECT e.last_name, d.department_name, l.city
FROM employees e, departments d, locations l
WHERE e.department_id = d.department_id
AND d.location_id = l.location_id;
```

Таблица может быть соединена и сама с собой. Например, данный запрос даст нам список отдела 90 с указанием имен начальников:

```
SELECT e.last_name, b.last_name boss
FROM employees e, employees b
WHERE e.department_id = 90
AND e.manager_id = b.employee_id;
```

Обратите внимание, что в отделе 90 работает больше людей, чем попало в предыдущую выборку:

```
SELECT last_name, manager_id FROM employees
WHERE department_id = 90;
```

Произошло это потому, что у сотрудника King нет начальника (manager_id is NULL). Если необходимо получить список всех сотрудников с заполненным указанием начальника, если он присутствует, используется внешнее соединение. Оператором внешнего соединения является знак (+):

```
SELECT таблица1.столбец, таблица2.столбец
FROM таблица1, таблица2
WHERE таблица1.столбец(+) = таблица2.столбец;
```

или

```
SELECT таблица1.столбец, таблица2.столбец
```

```
FROM таблица1, таблица2
WHERE таблица1.столбец = таблица2.столбец(+);
```

Предыдущий запрос можно переписать с использованием внешнего соединения:

```
SELECT e.last_name, b.last_name boss
FROM employees e, employees b
WHERE e.department_id = 90
  and e.manager_id = b.employee_id (+);
```

Другой вариант использования внешнего соединения. Обратите внимание на расположение оператора (+): он указывает правое или левое внешнее соединение будет использоваться, т.е. запрос будет дополняться пустыми данным из таблицы левого или правого столбцов условия эквисоединения.

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e, departments d
WHERE e.department_id (+) = d.department_id;
```

Существует альтернативный синтаксис задания соединений, определенный стандартом SQL:1999:

```
SELECT таблица1.столбец, таблица2.столбец
FROM таблица1
[CROSS JOIN таблица2] |
[NATURAL JOIN таблица2] |
[JOIN таблица2 USING (имя_столбца)] |
[JOIN таблица2
ON(таблица1.имя_столбца = таблица2.имя_столбца)] |
[LEFT|RIGHT|FULL OUTER JOIN таблица2
ON (таблица1.имя_столбца = таблица2.имя_столбца)];
```

Декартово произведение можно получить при помощи операции CROSS JOIN:

```
SELECT last_name, department_name
FROM employees
CROSS JOIN departments;
```

Операция NATURAL JOIN позволяет соединить таблицы по внешнему ключу, если имена столбцов первичного и внешнего ключа совпадают:

```
SELECT department_id, department_name, location_id, city
FROM departments
NATURAL JOIN locations;
```

Для определения произвольных соединений используется конструкция JOIN ON:

```
SELECT e.employee_id, e.last_name, e.department_id,
d.department_id, d.location_id
FROM employees e JOIN departments d
ON (e.department_id = d.department_id);
```

При помощи этой конструкции можно соединять произвольное количество таблиц:

```
SELECT employee_id, city, department_name
FROM employees e
JOIN departments d
ON d.department_id = e.department_id
JOIN locations l
ON d.location_id = l.location_id;
```

В соответствие со стандартом SQL: 1999 внутреннее соединение – это соединение двух таблиц, возвращающее только соответствующие условию соединения строки.

Соединение двух таблиц, возвращающее как строки внутреннего соединения, так и несоответствующие строки левой (правой) таблицы, – это левое (правое) внешнее соединение.

Полное внешнее соединение возвращает результаты внутреннего соединения, а также левого и правого внешнего соединений.

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e
LEFT OUTER JOIN departments d
ON (e.department_id = d.department_id);
```

Пример левого, правого и полного внешнего соединения, оцените разницу в полученных результатах:

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e
LEFT OUTER JOIN departments d
ON (e.department_id = d.department_id);

SELECT e.last_name, e.department_id, d.department_name
FROM employees e
RIGHT OUTER JOIN departments d
ON (e.department_id = d.department_id);

SELECT e.last_name, e.department_id, d.department_name
FROM employees e
FULL OUTER JOIN departments d
ON (e.department_id = d.department_id);
```

Синтаксис SQL:1999 позволяет формулировать и дополнительные условия на соединение:

```
SELECT e.employee_id, e.last_name, e.department_id,
d.department_id, d.location_id
FROM employees e JOIN departments d
ON (e.department_id = d.department_id)
AND e.manager_id = 149;
```

Задания для самостоятельной работы.

1. Напишите запрос для вывода фамилии, номера отдела и названия отдела для всех служащих.
2. Выведите список всех должностей в отделе 80 (должности в списке не должны повторяться) и местоположений отдела.
3. Напишите запрос для вывода фамилии, названия отдела, идентификатора местоположения отдела и города, в котором он находится, для всех служащих, зарабатывающих комиссионные.
4. Выведите фамилии служащих, содержащие букву "a" (в строчном регистре), с названиями отделов.
5. Напишите запрос для вывода фамилии, должности, номера отдела и названия отдела всех служащих, работающих в городе Toronto.
6. Выведите фамилии и номера служащих вместе с фамилиями и номерами их менеджеров. Назовите столбцы Employee, Emp#, Manager и Mgr#.
7. Измените предыдущий запрос так, чтобы получить фамилии всех служащих, включая King, который не имеет менеджера. Упорядочьте результат по возрастанию номера служащего.
8. Создайте запрос для вывода номера отдела, фамилии служащего и фамилий всех служащих, работающих в одном отделе с данным служащим. Дайте столбцам соответствующие имена.
9. Создайте запрос для вывода фамилий и дат найма всех служащих, нанятых после Davies.
10. По всем служащим, нанятым раньше своих менеджеров, выведите фамилии и даты найма самих служащих, а также фамилии и даты найма их менеджеров. Назовите столбцы Employee, Emp Hired, Manager и Mgr Hired.

Лабораторная работа №8. Агрегирующие функции, раздел GROUP BY.

Цель работы: Научиться использованию агрегирующих функций и группировке строк при помощи GROUP BY.

Теоретическая часть.

Как уже говорилось, различают однострочные и многострочные функции. Однострочные были рассмотрены ранее, многострочные же (или групповые, или агрегирующие) функции – это такие функции, которые работают с группой строк и возвращают один результат на группу. Например, такой запрос вычислит групповую функцию MAX (максимум) для столбца SALARY над всеми строками таблицы EMPLOYEES, т.е. выдаст максимальную зарплату среди всех сотрудников:

```
SELECT max(salary) FROM employees;
```

Типы групповых функций:

- AVG – среднее значение
- COUNT - количество
- MAX – максимальное значение
- MIN – минимальное значение
- STDDEV – стандартное отклонение
- SUM - сумма
- VARIANCE – дисперсия

Общий синтаксис использования групповых функций:

```
SELECT [столбец,] групп_функция(столбец), ...
FROM таблица
[WHERE условие]
[GROUP BY столбец]
[ORDER BY столбец];
```

Функции AVG и SUM применяются к столбцам с числовыми данными:

```
SELECT AVG(salary), MAX(salary),
MIN(salary), SUM(salary)
FROM employees
WHERE job_id LIKE '%REP%';
```

Функции MAX и MIN применяются к данным любого типа:

```
SELECT MIN(hire_date), MAX(hire_date)
FROM employees;
```

COUNT(*) возвращает количество строк в выборке:

```
SELECT COUNT(*)
FROM employees
WHERE department_id = 50;
```

COUNT(выражение) возвращает количество строк с определенными значениями (не NULL) для выражения. Например, количество сотрудников отдела 80, получающих комиссионные:

```
SELECT COUNT(commission_pct)
FROM employees
WHERE department_id = 80;
```

COUNT(DISTINCT выражение) возвращает количество уникальных определенных значений выражения. Например, количество уникальных номеров отделов из таблицы EMPLOYEES:

```
SELECT COUNT(DISTINCT department_id) FROM employees;
```

Групповые функции игнорируют неопределенные значения в столбцах:

```
SELECT AVG(commission_pct)
FROM employees;
```

Функция NVL заставляет групповые функции включать неопределенные значения:

```
SELECT AVG(NVL(commission_pct, 0))
FROM employees;
```

Предложение GROUP BY разбивает строки таблицы на группы, по которым вычисляются функции. Все столбцы, которые входят в список SELECT и к которым не применяются групповые функции, должны быть указаны в предложении GROUP BY. Например, если необходимо посчитать среднюю зарплату по каждому отделу:

```
SELECT department_id, AVG(salary) FROM employees
GROUP BY department_id;
```

Столбец, указанный в GROUP BY, может отсутствовать в списке SELECT:

```
SELECT AVG(salary)
FROM employees
GROUP BY department_id;
```

Группировать можно по нескольким столбцам:

```
SELECT department_id dept_id, job_id, SUM(salary)
FROM employees
GROUP BY department_id, job_id
ORDER BY sum(salary) desc;
```

Предложение WHERE для исключения групп не используется, нельзя использовать групповые функции в предложении WHERE. Для исключения некоторых групп следует пользоваться предложением HAVING:

```
-- Ошибочный запрос
SELECT department_id, AVG(salary)
FROM employees
WHERE AVG(salary) > 8000
GROUP BY department_id;

-- Верный запрос
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id
HAVING AVG(salary) > 8000;
```

Групповые функции могут быть вложенными:

```
SELECT MAX(AVG(salary))
FROM employees
GROUP BY department_id;
```

Задания для самостоятельной работы.

1. Напишите запрос для вывода самого высокого, самого низкого и среднего оклада по всем служащим, а также суммы всех окладов. Назовите столбцы Maximum, Minimum, Average и Sum. Округлите суммы до ближайшего целого значения.
2. Измените предыдущий запрос так, чтобы получить самый низкий, самый высокий и средний оклады, а также сумму окладов отдельно по каждой должности.
3. Напишите запрос для вывода должности и количества служащих, занимающих каждую должность.
4. Получите количество служащих, имеющих подчиненных, без их перечисления. Назовите столбец Number of Managers. Подсказка: используйте столбец MANAGER_ID для определения числа менеджеров.
5. Напишите запрос для вывода разности между самым высоким и самым низким окладами. Назовите столбец DIFFERENCE.
6. Напишите запрос для вывода номера каждого менеджера, имеющего подчиненных, и заработную плату самого низкооплачиваемого из его подчиненных. Исключите менеджеров, для которых неизвестны их менеджеры. Исключите все группы, где минимальный оклад составляет менее $6000. Отсортируйте выходные строки в порядке убывания оклада.
7. Напишите запрос для вывода названия отдела, местоположения отдела, количества служащих и среднего оклада по этому отделу. Назовите столбцы Name,

Location, Number of People и Salary. Округлите средний оклад до ближайшего целого значения.

8. Напишите запрос для вывода общего количества служащих и количества служащих, нанятых в 1995, 1996, 1997 и 1998 годах. Дайте соответствующие заголовки столбцам.

Лабораторная работа №9. Подзапросы.

Цель работы: Понять проблемы, решаемые подзапросами. Изучить типы подзапросов. Научиться использовать однострочные и многострочные подзапросы.

Теоретическая часть.

Иногда для ответа на главный вопрос необходимо сначала ответить на другой, более частный вопрос. Например, для того, чтобы ответить на вопрос "У кого зарплата больше, чем у сотрудника по фамилии Austin?" надо сначала ответить на вопрос "А какая зарплата у Austin?". Можно последовательно выполнить два запроса:

```
SELECT salary
FROM employees
WHERE last_name = 'Austin';
-- Получим 4800 и выполним второй запрос:
SELECT last_name
FROM employees
WHERE salary > 4800;
```

Для того, чтобы оформить это одним SQL предложением, можно использовать подзапрос:

```
SELECT last_name
FROM employees
WHERE salary >
     (SELECT salary
     FROM employees
     WHERE last_name = 'Austin');
```

Общий синтаксис использования подзапросов:

```
SELECT список_выбора
FROM таблица
WHERE выражение оператор
     (SELECT список_выбора
     FROM таблица);
```

Указания по использованию подзапросов:

- Подзапрос должен быть заключен в скобки.
- Подзапрос должен находиться справа от оператора сравнения.
- В подзапросе не следует использовать предложение ORDER BY, если только не применяется “TOP-N” анализ.
- В однострочных подзапросах используются однострочные операторы.
- В многострочных подзапросах используются многострочные операторы.

Различают однострочные (возвращающие ровно одну строку) и многострочные (возвращающие ноль или более строк) подзапросы.

Для однострочных подзапросов используется обычные однострочные операторы сравнения (=, >, <, <>, >=, <=):

```
SELECT last_name, job_id, salary
FROM employees
WHERE job_id =
     (SELECT job_id
     FROM employees
     WHERE employee_id = 141)
AND salary >
     (SELECT salary
     FROM employees
     WHERE employee_id = 143);
```

В подзапросах можно использовать групповые функции, подзапрос с групповой функцией без GROUP BY – однострочный:

```
SELECT last_name, job_id, salary
FROM employees
WHERE salary =
     (SELECT MIN(salary)
     FROM employees);
```

Использование однострочных операторов с многострочными или пустыми запросами запрещено:

```
SELECT employee_id, last_name
FROM employees
WHERE salary =
     (SELECT MIN(salary)
     FROM employees
     GROUP BY department_id);

SELECT last_name, job_id
FROM employees
WHERE job_id =
     (SELECT job_id
     FROM employees
     WHERE last_name = 'Haas');
```

Для многострочных подзапросов используются специальные многострочные операторы сравнения:

- IN – равно какому-либо значению из подзапроса.
- ANY – сравнивает значение с каким-либо значением из подзапроса.
- ALL – сравнивает значение со всеми значениями из подзапроса.

Использование IN:

```
SELECT last_name FROM employees
WHERE department_id in
     (SELECT department_id
     FROM departments WHERE location_id = 1700);
```

Использование ANY (список сотрудников, для которых есть кто-то из программистов, кто получает больше):

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary < ANY
     (SELECT salary
     FROM employees
     WHERE job_id = 'IT_PROG')
AND job_id <> 'IT_PROG';
```

Использование ALL (список сотрудников, получающих меньше, чем любой из программистов):

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary < ALL
     (SELECT salary
     FROM employees
     WHERE job_id = 'IT_PROG')
AND job_id <> 'IT_PROG';
```

Обратите внимание, что сравнения используется обычный оператор, поэтому и неопределенные значения обрабатываются штатным образом:

```
SELECT count(*) FROM employees;

SELECT count(*) FROM employees
WHERE manager_id in
     (SELECT manager_id FROM employees);
```

Кроме рассмотренных выше операторов возможно использование еще одного оператора [NOT] EXISTS. Оператор EXISTS проверяет существование строк в наборе результатов подзапроса: если в результате выполнения подзапроса есть хотя бы одна строка, не важно с какими значениями, условие отмечается как верное, если нет ни одной строки – как ложное. Например:

```
SELECT employee_id, last_name, job_id, department_id
FROM employees outer
WHERE EXISTS (SELECT 'X'
```

```
                    FROM employees
                    WHERE manager_id =
                    outer.employee_id);
```

Рассмотрим еще несколько вариантов использования подзапросов.

Подзапрос может использовать в качестве одной из таблиц в классе FROM, например, можно использовать следующий запрос для получения списка сотрудников, получающих зарплату больше, чем в среднем по отделу:

```
SELECT a.last_name, a.salary,
a.department_id, b.salavg
FROM employees a, (SELECT department_id,
                          AVG(salary) salavg
                   FROM employees
                   GROUP BY department_id) b
WHERE a.department_id = b.department_id
AND a.salary > b.salavg;
```

Выделяют специальный тип подзапросов – скалярные. Это такой подзапрос, который возвращает ровно одно значение (одна строка из одного столбца). Таки подзапросы можно использовать в качестве значений в любом классе команды SELECT, кроме класса GROUP BY. Например:

```
SELECT last_name,
      (SELECT department_name
       FROM departments s
       WHERE s.department_id=m.department_id) dep_name
FROM employees m
WHERE last_name like 'K%';

SELECT employee_id, last_name
FROM employees e
ORDER BY
(SELECT department_name
FROM departments d
WHERE e.department_id = d.department_id);
```

Задания для самостоятельной работы.

1. Создайте запрос для вывода фамилии и даты найма каждого служащего, работающего в одном отделе с Zlotkey. Исключите Zlotkey из выходных данных.
2. Создайте запрос для вывода номеров и фамилий всех служащих, оклад которых выше среднего. Отсортируйте выходные данные в порядке увеличения окладов.
3. Создайте запрос для вывода номеров и фамилий всех служащих, работающих в одном отделе с любым служащим, фамилия которого содержит букву "u".

4. Создайте запрос для вывода фамилии, номера отдела и должности каждого служащего, идентификатор местоположения отдела которого равен 1700.
5. Получите список фамилий и окладов всех служащих, подчиненных Кингу (King).
6. Получите номер отдела, фамилию и должность для каждого служащего, работающего в администрации (department_name = 'Executive').
7. Создайте запрос для вывода фамилии, номера отдела и оклада всех служащих, чей номер отдела и оклад совпадают с номером отдела и окладом любого служащего, зарабатывающего комиссионные.
8. Выведите фамилию, название отдела и оклад всех служащих, чей оклад и комиссионные совпадают с окладом и комиссионными любого служащего, работающего в отделе, местоположение которого (location_ID) 1700.
9. Создайте запрос для вывода фамилии, даты найма и оклада всех служащих, которые получают такой же оклад и такие же комиссионные, как Kochhar.
10. Выведите фамилию, должность и оклад всех служащих, оклад которых превышает оклад каждого клерка торгового менеджера (JOB_ID = 'SA_MAN'). Отсортируйте результаты по убыванию окладов.
11. Выведите номера, фамилии и отделы служащих, живущих в городах, названия которых начинаются с буквы "T".
12. Напишите запрос для нахождения всех сотрудников, которые зарабатывают больше среднего оклада по их отделу. Выведите фамилию, оклад, номер отдела и средний оклад по отделу. Отсортируйте результаты по средней зарплате. Используйте псевдонимы для выбираемых столбцов, приведенные в приводимом ниже примере выходных результатах.
13. Найдите всех сотрудников, не являющихся руководителями.
14. Выведите фамилии сотрудников, зарабатывающих меньше среднего оклада по их отделу.
15. Выведите фамилии сотрудников, у которых есть коллеги по отделу, которые были приняты на работу позже, но имеют более высокий оклад.
16. Выведите номера, фамилии и наименования отделов всех сотрудников. Примечание: используйте скалярный подзапрос в команде SELECT для вывода наименований отделов.

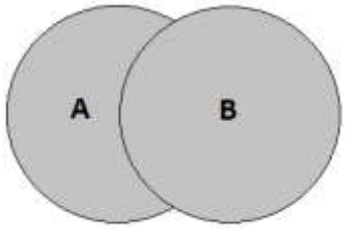
Лабораторная работа №10. Операция над множествами.

Цель работы: Изучить операции UNION, INTERSECT, MINUS. Научиться применять сортировку при операциях над множествами.

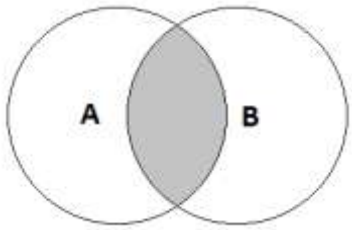
Теоретическая часть.

В SQL определены три операции над множествами (результатами запросов):

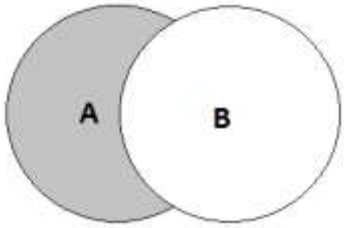
A UNION **B** (объединение, возвращает результаты обоих запросов после устранения дубликатов):



A INTERSECT **B** (пересечение, возвращает только те строки, которые входят в результаты обоих запросов):



A MINUS **B** (вычитание, возвращает строки из первого запроса, за исключением тех, которые входят во второй запрос):



Еще есть операция UNION ALL, которая работает так же как UNION, но при этом не удаляет из полученного результата повторяющиеся записи.

Например, данный запрос отображает информацию о текущем и прошлом месте работы для всех служащих. Каждое место работы отображается только один раз:

```
SELECT employee_id, job_id
FROM employees
UNION
SELECT employee_id, job_id
```

```
FROM job_history;
```

Те же запросы, но с использованием UNION ALL дадут немного большую выборку, т.к. дубли не будут удалены:

```
SELECT employee_id, job_id
FROM employees
UNION ALL
SELECT employee_id, job_id
FROM job_history;
```

Данных запрос отображает номера служащих, менявших должность за время карьеры:

```
SELECT employee_id
FROM employees
INTERSECT
SELECT employee_id
FROM job_history;
```

А данный запрос отображает номера служащих, ни разу не менявших должность:

```
SELECT employee_id
FROM employees
MINUS
SELECT employee_id
FROM job_history;
```

Ограничения на операторы работы с множествами:

- Количество и тип данных в выражениях SELECT должны совпадать.
- Для изменения последовательности выполнения можно использовать скобки.
- Выражение ORDER BY:
    - Может использоваться только в конце выражения
    - Принимает имя столбца, псевдонимы из первого выражения SELECT, или позиционное перечисление.

Пример использования ORDER BY:

```
SELECT department_id,job_id FROM employees
UNION
SELECT department_id,job_id FROM job_history
ORDER BY department_id;
```

Задания для самостоятельной работы.

1. Используя операторы работы с множествами, выведите номера отделов (department_id), в которых нет служащих с идентификатором должности (job_id) ST_CLERK.
2. Используя операторы работы с множествами, выведите идентификаторы и наименования стран, в которых не располагаются отделы компании.
3. Используя операторы работы с множествами, выведите список должностей отделов 10, 50 и 20 в таком же порядке отделов. Выведите столбцы job_id и department_id.
4. Выведите номер сотрудника и идентификатор его должности, если его текущая должность совпадает с той, которую он уже занимал, работая в компании.
5. Напишите составной запрос, который выводит следующее: имена и отделы всех сотрудников из таблицы EMPLOYEES, независимо от того, относятся ли они к какому-то отделу или нет; номера и наименования всех отделов из таблицы DEPARTMENTS, независимо от того, есть ли в них сотрудники или нет.

Лабораторная работа №11. Вставка, модификация и удаление данных.

Цель работы: Научиться использованию операторов INSERT, UPDATE и DELETE для вставки, модификации и удаления данных.

Теоретическая часть.

Перед началом работы выполните эти команды для создания дополнительных учебных таблиц:

```
CREATE TABLE sales_reps
(id number, name varchar2(50),
 salary number, commission_pct number);

CREATE TABLE copy_emp
as SELECT * FROM employees
   WHERE mod(employee_id,10)=0;

CREATE TABLE copy_dep
as SELECT * FROM departments
   WHERE department_id not in (70,100,30);
```

Язык манипулирования данными (DML) используется для:

- Вставки новых данных в таблицы.
- Изменения существующих данных в таблицах.
- Удаления существующих данных из таблиц.

Транзакция – это совокупность команд DML, образующих логическую единицу работы (либо все внесенные этими командами изменения принимаются, либо все отменяются).

Для вставки строк в таблицу используется команда INSERT:

```
INSERT INTO таблица [(столбец [, столбец...])]
VALUES (значение [, значение...]);
```

Команда INSERT позволяет явно перечислить все столбцы и указать, какими значениями следует их заполнить, либо использовать порядок столбцов по умолчанию:

```
INSERT INTO copy_dep
(department_id, department_name, manager_id, location_id)
VALUES (70, 'Public Relations', 100, 1700);
```

или

```
INSERT INTO copy_dep
VALUES (100, 'Finance', null, null);
```

или

```
INSERT INTO copy_dep (department_id, department_name)
VALUES (30, 'Purchasing');
```

Рекомендуется всегда явно перечислять столбцы таблицы при вставке – это позволяет улучшить читаемость SQL предложения и не зависеть от конкретного порядка столбцов в таблице, который, со временем, может измениться.

В классе VALUES можно указывать вызовы функций:

```
INSERT INTO copy_emp (employee_id,
  first_name, last_name,
  email, phone_number,
  hire_date, job_id, salary,
  commission_pct, manager_id,
  department_id)
VALUES (113,
  'Louis', 'Popp',
  'LPOPP', '515.124.4567',
  SYSDATE, 'AC_ACCOUNT', 6900,
  NULL, 205, 100);
```

Для копирования строк из другой таблицы или просто вставке нескольких строк одновременно, можно использовать подзапрос в команде INSERT, при этом класс VALUES не используется, а количество столбцов, указанных в INSERT, должно совпадать с количеством столбцов в подзапросе.

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
SELECT employee_id, last_name, salary, commission_pct
       FROM employees
       WHERE job_id LIKE '%REP%';
```

Для изменения данных используется команда UPDATE:

```
UPDATE таблица
SET столбец = значение [, столбец = значение, ...]
[WHERE условие];
```

Класс SET указывает, какие столбцы надо изменить и на что, а класс WHERE указывает, на какую строку или строки будет распространяться это изменение:

```
UPDATE copy_emp
SET department_id = 70
WHERE employee_id = 113;
```

Если предложение WHERE отсутствует, обновляются все строки таблицы.

Одновременно можно обновлять 2 и более столбцов, больше того, для вычисления значений можно использовать подзапросы. В данном примере происходит изменение должности и оклада служащего под номером 114, чтобы они стали такими же, как у служащего под номером 205:

```
UPDATE copy_emp
SET job_id = (SELECT job_id
              FROM employees
              WHERE employee_id = 205),
salary = (SELECT salary
          FROM employees
          WHERE employee_id = 205)
          WHERE employee_id = 110;
```

Для изменения строк таблицы на основе значений из другой таблицы используйте подзапросы в командах UPDATE:

```
UPDATE copy_emp
SET department_id = (SELECT department_id
                     FROM employees
                     WHERE employee_id = 100)
WHERE job_id = (SELECT job_id
                FROM employees
                WHERE employee_id = 200);
```

Для удаления строк из таблицы используется команда DELETE:

```
DELETE [FROM] таблица
[WHERE условие];
```

Конкретная строка или строки удаляются с помощью предложения WHERE:

```
DELETE FROM copy_dep
WHERE department_name = 'Finance';
```

Если предложение WHERE отсутствует, удаляются все строки таблицы.

Для удаления строк на основе значений из другой таблицы используйте подзапросы в командах DELETE:

```
DELETE FROM copy_emp
WHERE department_id =
(SELECT department_id
 FROM departments
 WHERE department_name LIKE '%Public%');
```

Если при создании таблицы было указано значение по умолчанию для столбца, то стандарт SQL:1999 позволяет использовать в командах INSERT и UPDATE ключевое слово DEFAULT там, где указывается значение столбца:

```
INSERT INTO copy_dep
(department_id, department_name, manager_id)
VALUES (300, 'Engineering', DEFAULT);

UPDATE copy_dep
SET manager_id = DEFAULT WHERE department_id = 10;
```

Команда MERGE позволяет внести изменения в таблицу, если строка с таким ключом уже есть в таблице, или же добавить новую строку, в противном случае:

```
MERGE INTO имя_таблицы псевдоним_таблицы
USING (таблица|представление|подзапрос) псевдоним
ON (условие_соединения)
WHEN MATCHED THEN
UPDATE SET
столбец1 = значение_столбца1,
столбец2 = значение_столбца2
WHEN NOT MATCHED THEN
INSERT (список_столбцов)
VALUES (список_столбцов);
```

Вставка или изменение строк таблицы COPY_EMP для установления соответствия с значениями таблицы EMPLOYEES:

```
MERGE INTO copy_emp c
USING employees e
ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN
  UPDATE SET
    c.first_name = e.first_name,
    c.last_name = e.last_name,
    c.email = e.email,
    c.phone_number = e.phone_number,
    c.hire_date = e.hire_date,
    c.job_id = e.job_id,
    c.salary = e.salary,
    c. commission_pct = e.commission_pct,
    c.manager_id = e.manager_id,
    c.department_id = e.department_id
WHEN NOT MATCHED THEN
  INSERT VALUES(e.employee_id, e.first_name, e.last_name,
    e.email, e.phone_number, e.hire_date, e.job_id,
```

```
        e.salary, e.commission_pct, e.manager_id,
        e.department_id);
```

После выполнения DML команд рекомендуется в явном виде завершить транзакцию оператором COMMIT (фиксация транзакции) или ROLLBACK (откат транзакции).

Задания для самостоятельной работы.

1. Выполните sql-предложение для создания таблицы MY_EMPLOYEE , которая будет использоваться для упражнений:

```
CREATE TABLE my_employee
(id NUMBER(4) CONSTRAINT my_employee_id_nn NOT NULL,
 last_name VARCHAR2(25),
 first_name VARCHAR2(25),
 userid VARCHAR2(8),
 salary NUMBER(9,2));
```

2. Вставьте в таблицу MY_EMPLOYEE первую строку из нижеприведенных образцов. Не указывайте столбцы в предложении INSERT.
3. Вставьте в таблицу MY_EMPLOYEE вторую строку из вышеуказанных образцов. На этот раз укажите столбцы явно в предложении INSERT.
4. Убедитесь в том, что данные вставлены в таблицу.
5. Сделайте изменения постоянными.
6. Смените фамилию служащего номер 3 на "Drexler".
7. Сделайте оклад равным 1000 для всех служащих, имеющих оклад ниже 900.
8. Проверьте изменения в таблице.
9. Удалите из таблицы MY_EMPLOYEE строку для служащей Betty Dancs.
10. Проверьте изменения в таблице.
11. Зафиксируйте все незафиксированные изменения.

Лабораторная работа №12. Транзакции.

Цель работы: Научиться использованию операторов COMMIT, ROLLBACK, ROLLBACK TO SAVEPOINT. Изучить уровни изолированности транзакций, понять причины возникновения ситуации Deadlock.

Теоретическая часть.

Транзакция базы данных начинается выполнением любого SQL предложения и заканчивается одним из событий:

- Выполнение команды COMMIT или ROLLBACK;
- Выполнения команды DDL (автоматическая фиксация);
- Выход пользователя из текущей сессии;
- Отказ системы (автоматический откат).

Команда COMMIT фиксирует все изменения в данных, осуществленные транзакцией.

Команда ROLLBACK отменяет все изменения в данных, осуществленные транзакцией, возвращая базу данных к тому состоянию, которое она имела в момент начала транзакции.

Внутри транзакции можно определить дополнительные точки сохранения – SAVEPOINT. В этом случае откат можно осуществлять не только к самому началу транзакции, но и к любой из определенных точек сохранения:

```
COMMIT;
-- Начало новой транзакции
SELECT count(*) FROM copy_emp WHERE department_id = 10;
UPDATE copy_emp SET department_id = 10
WHERE employee_id = 110;
-- Устанавливаем точку сохранения с именем upd_done;
SAVEPOINT upd_done;
SELECT count(*) FROM copy_emp WHERE department_id = 10;
INSERT INTO copy_emp (employee_id,last_name,email,salary,
   department_id,hire_date,job_id)
VALUES (1001,'Tompson','TMP',1000,
   10,sysdate,'SA_REP');
SELECT count(*) FROM copy_emp WHERE department_id = 10;
-- Откат до точки upd_done
ROLLBACK TO upd_done;
SELECT count(*) FROM copy_emp WHERE department_id = 10;
ROLLBACK;
SELECT count(*) FROM copy_emp WHERE department_id = 10;
```

Состояние данных до выполнения в транзакции команды COMMIT или ROLLBACK:

- Предыдущее состояние данных может быть восстановлено.

- Текущий пользователь может просмотреть результаты своих операций DML с помощью команды SELECT.
- Другие пользователи не могут видеть результаты команд DML, выполняемых текущим пользователем.
- Измененяемые строки блокируются, и другие пользователи не могут обновлять их содержимое.

Состояние данных после выполнения команды COMMIT:

- Измененные данные записываются в базу данных.
- Предшествующее состояние данных теряется.
- Все пользователи могут видеть результаты.
- Измененные строки разблокируются, и другие пользователи получают доступ к ним для обработки данных.
- Все точки сохранения стираются.

Состояние данных после выполнения команды ROLLBACK:

- Изменения в данных отменяются.
- Данные возвращаются в прежнее состояние.
- Блокировка строк, над которыми выполнялись операции, отменяется.

Если в процессе исполнения команды DML произошла ошибка, то автоматически отменяются все изменения данных, которые были внесены этой командой до момента возникновения ошибки (используется неявная точка сохранения, которая устанавливается сервером перед началом каждой транзакции). Все прочие изменения в транзакции (сделанные до момента выполнения DML с ошибкой) сохраняются и транзакция должна быть завершена командой COMMIT или ROLLBACK.

В момент изменения данных одним пользователем эти данные блокируются для изменения другими пользователями до тех пор, пока в изменяющей транзакции не будет выполнена фиксация или откат.

Для выполнения следующих примеров откройте два разных соединения с сервером БД (запустите два экземпляра своего клиента). Назовем их сессиями A и B.

В сессии A изменим одну строку:

```
UPDATE copy_emp SET last_name = 'Newman'
WHERE employee_id = 100;
```

В сессии B эти изменения еще не видны, т.к. сессия A еще не зафиксировала транзакцию:

```
SELECT last_name FROM copy_emp
WHERE employee_id = 100;
```

Но изменить эти же данные в сессии B невозможно, команда UPDATE становится в ожидание завершения транзакции в сессии A:

```
UPDATE copy_emp SET last_name = 'Oldman'
WHERE employee_id = 100;
```

В сессии A завершим транзакцию:

```
COMMIT;
```

И сразу видим, что выполнилась и команда в сессии B, завершим транзакцию и в этой сессии, но откатом:

```
ROLLBACK;
```

Убедимся, что после того, как все транзакции завершены, в обеих сессиях мы видим одинаковые данные:

```
SELECT last_name FROM copy_emp WHERE employee_id = 100;
```

К сожалению, не все блокировки можно согласовать простым ожиданием. Возможны ситуации так называемых "мертвых блокировок" (deadlock), в которые приходится вмешиваться серверу. Например:

В сессии A обновляем одну строку:

```
UPDATE copy_emp SET last_name = 'King'
WHERE employee_id = 100;
```

В сессии B обновляем другую строку:

```
UPDATE copy_emp SET last_name = 'Queen'
WHERE employee_id = 200;
```

Это не запрещено, т.к. Oracle может управлять блокировками на уровне строк и разные сессии могут одновременно изменять разные строки одной и той же таблицы.

Теперь в сессии A попробуем изменить тут запись, которая заблокирована сессией B:

```
UPDATE copy_emp SET salary = 45000
WHERE employee_id = 200;
```

Несмотря на то, что мы меняем другой столбец, заблокирована строка целиком и сессия впадает в состояния ожидания завершения транзакции в сессии B. В это время в сессии B мы пробуем изменить ту строку, которая заблокирована сессией A. Т.е. делаем перекрестные блокировки: A будет ждать B, а B будет ждать A. И это ожидание само

никогда не кончится, поэтому сервер распознает такие ситуации и автоматически прерывает одну из операций.

Выполним в сессии B:

```
UPDATE copy_emp SET salary = 35000
WHERE employee_id = 100;
```

И увидим, что одна из операций была прервана сервером с ошибкой "deadlock detected while waiting for resource".

Т.к. была прервана только одна операция, а не вся транзакция целиком, надо корректно завершить обе транзакции, например, командой COMMIT в обеих сессиях:

```
COMMIT;
```

Задания для самостоятельной работы.

1. Вставьте в таблицу copy_emp какую-либо строку строку.
2. Проверьте вставку данных в таблицу.
3. Создайте точку сохранения в ходе транзакции.
4. Удалите все данные из таблицы.
5. Убедитесь в том, что таблица пуста.
6. Отмените последнюю операцию DELETE, не отменяя предыдущую операцию INSERT.
7. Убедитесь в том, что вставленная строка присутствует в таблице.
8. Сделайте добавление данных постоянным.

Лабораторная работа №13. DDL. Определение таблиц.

Цель работы: Научиться создать простейшие таблицы, модифицировать таблицы, определять ограничения PRIMARY/FOREIGN KEY и CHECK CONSTRAINTS.

Теоретическая часть.

Правила формирования имен таблиц и столбцов:

- Должны начинаться с буквы
- Могут быть длиной от 1 до 30 символов
- Должны содержать только символы A–Z, a–z, 0–9, _, $ и #
- Не должны совпадать с именем другого объекта, принадлежащего этому же самому пользователю
- Не должны совпадать со словом, зарезервированным сервером Oracle

Для создания новой таблицы используется команда CREATE TABLE в которой задается имя таблицы и имена, типы данных, размер и значения по умолчанию для столбцов:

```
CREATE TABLE [схема.]таблица
(столбец тип_данных [DEFAULT выражение][, ...]);
```

Опция DEFAULT (значение по умолчанию):

- Задает значение по умолчанию, если при вставке данных значение не указано явно.
- В качестве значения допускается литерал, выражение или функция SQL.
- Не может использоваться имя другого столбца или псевдостолбец.
- Тип данных, используемый по умолчанию, должен совпадать с типом данных столбца.

Например:

```
CREATE TABLE dept
(deptno NUMBER(2),
dname VARCHAR2(14),
loc NUMBER(4) DEFAULT 1000);
```

Для определения столбцов можно использовать следующие типы данных:

Тип данных	Описание
VARCHAR2(размер)	Символьные данные переменной длины
CHAR(размер)	Символьные данные постоянной длины
NUMBER(p,s)	Числовые данные переменной длины
DATE	Значения даты и времени
TIMESTAMP	Дата до долей секунды
TIMESTAMP WITH TIME ZONE	Вариант типа данных TIMESTAMP, который содержит смещение для временной зоны
TIMESTAMP WITH	Вариант типа данных TIMESTAMP, который содержит

LOCAL TIME ZONE	смещение для временной зоны, связанное с временной зоной базы данных
INTERVAL YEAR TO MONTH	Интервал времени в годах и месяцах
INTERVAL DAY TO SECOND	Интервал времени в днях с точностью до минут и секунд
LONG	Символьные данные переменной длины до 2 гигабайтов
CLOB	Символьные данные до 4 гигабайтов
RAW и LONG RAW	Необработанные (“сырые“) двоичные данные
BLOB	Двоичные данные до 4 гигабайтов
BFILE	Двоичные данные, которые хранятся во внешнем файле; длина до 4 гигабайтов
ROWID	Уникальный адрес строки в 64-ой символьной системе кодировки

Таблицы можно создавать сразу с заполнением данными с использование подзапроса. Количество заданных столбцов должно совпадать с количеством столбцов в подзапросе. Столбцы могут быть определены с именами и значениями, используемыми по умолчанию:

```
CREATE TABLE table
[(столбец, столбец...)]
AS подзапрос;
```

Например:

```
CREATE TABLE dept80
AS
SELECT employee_id, last_name,
salary*12 ANNSAL,
hire_date
FROM employees
WHERE department_id = 80;
```

Для изменения определения таблицы используется команда ALTER TABLE. Команда используется для добавления, изменения или удаления столбцов:

```
ALTER TABLE таблица
ADD (столбец тип_данных [DEFAULT выражение]
[, столбец тип_данных]...);
```

```
ALTER TABLE таблица
MODIFY (столбец тип_данных [DEFAULT выражение]
[, столбец тип_данных]...);
```

```
ALTER TABLE таблица
DROP (столбец);
```

Например, добавление столбца

```
ALTER TABLE dept80
ADD (job_id VARCHAR2(9));
```

Изменение столбца:

```
ALTER TABLE dept80
MODIFY (job_id VARCHAR2(30) DEFAULT 'SA_REP');
```

Удаление столбца:

```
ALTER TABLE dept80
DROP COLUMN job_id;
```

Для удаления таблицы используется команда DROP TABLE, при этом удаляются все данные из таблицы, сама таблица и все связанные с ней объекты, такие как индексы и триггеры. Результат этой команды отменить невозможно. Например:

```
DROP TABLE dept80;
```

Для того, чтобы просто удалить все данные из таблицы, но не саму таблицу, можно использовать команду TRUNCATE TABLE (откат также невозможен, если нужно удалить с возможностью отката – используйте DML команду DELETE), например:

```
TRUNCATE TABLE dept;
```

Для переименования таблицы можно использовать команду RENAME:

```
RENAME dept TO detail_dept;
```

При создании и модификации таблиц можно задавать ограничения (CONSTRAINTS):

- Ограничения обеспечивают выполнение правил на уровне таблицы.
- Ограничения предотвращают удаление таблицы при наличии подчиненных данных в других таблицах.
- В Oracle допускаются следующие виды ограничений:
  - NOT NULL
  - UNIQUE
  - PRIMARY KEY
  - FOREIGN KEY
  - CHECK

Каждое ограничение имеет имя. Имя можно задать явно при создании, или оно будет присвоено автоматически в виде SYS_xxxxxxx. Рекомендуется задавать имена ограничений явно.

Ограничения могут задаваться как на уровне таблицы, так и на уровне конкретного столбца таблицы:

```
CREATE TABLE [схема.]таблица
(столбец тип_данных [DEFAULT выражение]
[ограничение_на_уровне_столбца],
...
[ограничение_на_уровне_таблицы][,...]);
```

Например:

```
CREATE TABLE emp2(
employee_id NUMBER(6),
last_name VARCHAR2(25) NOT NULL,
first_name VARCHAR2(20),
job_id VARCHAR2(10) CONSTRAINT emp2_job_id_nn NOT NULL,
salary number(2) CONSTRAINT emp2_sal_min CHECK (salary > 0),
email varchar2(30) CONSTRAINT emp2_email_uk UNIQUE,
department_id number(4),
manager_id number(6),
CONSTRAINT emp2_emp_id_pk PRIMARY KEY (EMPLOYEE_ID),
CONSTRAINT emp2_dept_fk FOREIGN KEY (department_id)
REFERENCES departments(department_id) ON DELETE SET NULL
);
```

Ограничение NOT NULL может быть задано только на уровне столбца и запрещает создание/обновление строк с неопределенным значением этого столбца. Имя ограничения может быть сгенерировано системой (столбец last_name) или задано явно (столбец job_id).

Ограничения PRIMARY KEY и UNIQUE могут быть задано на уровне столбца или таблицы и позволяют контролировать уникальность одного или совокупности столбцов. PRIMARY KEY, помимо контроля уникальности, позволяет организовать ссылку на таблицу из другой таблицы при помощи внешнего ключа - FOREIGN KEY.

Ограничение FOREIGN KEY также может быть задано как на уровне столбца, так и на уровне таблицы. При определении внешнего ключа можно использовать дополнительные опции, указывающие как поступать с записями данной таблицы при удалении записи, на которую они ссылаются, из родительской таблицы:

- ON DELETE CASCADE. Разрешает удаление в родительской таблице с одновременным удалением зависимых строк в подчиненной таблице.
- ON DELETE SET NULL. При удалении строк в родительской таблице преобразует зависимое значение внешнего ключа в неопределенное.

Ограничение CHECK может быть задано как на уровне столбца, так и на уровне и таблицы. Ограничение формулирует условие, которому должна соответствовать каждая строка. Условие может ссылаться на имена столбцов (только текущего, если задано на уровне столбца или любого, если на уровне таблицы) и использовать простые условные операторы. Использование функций в CHECK ограничениях запрещено.

Ограничения можно добавлять/удалять и после создания таблицы при помощи команды ALTER. Синтаксис самого определения ограничения при этом сохраняется.

Например, добавление ограничения:

```
ALTER TABLE emp2
ADD CONSTRAINT emp2_manager_fk
FOREIGN KEY(manager_id)
REFERENCES employees(employee_id);
```

Удаление ограничения:

```
ALTER TABLE emp2
DROP CONSTRAINT emp2_manager_fk;
```

Ограничения можно не удалять, а временно отключать, а потом включать обратно. Ключевое слово CASCADE говорит о том, что надо выключить не только сам PRIMARY KEY, но и связанные с ним FOREIGN KEY:

```
ALTER TABLE employees
DISABLE CONSTRAINT emp_emp_id_pk CASCADE;
-- Проверить, что именно было выключено:
SELECT constraint_name, status FROM user_constraints;
ALTER TABLE employees
ENABLE CONSTRAINT emp_emp_id_pk;
-- часть ограничений осталось выключенными:
SELECT constraint_name, table_name, status
FROM user_constraints
WHERE status = 'DISABLED';
-- включим их
ALTER TABLE employees  ENABLE CONSTRAINT emp_manager_fk;
ALTER TABLE departments ENABLE CONSTRAINT dept_mgr_fk;
ALTER TABLE job_history ENABLE CONSTRAINT jhist_emp_fk;
```

При удалении столбца из таблицы можно использовать опцию CASCADE CONSTRAINTS, которая удалит все связанные с этим столбцом ограничения:

```
ALTER TABLE emp2 DROP (employee_id, job_id) CASCADE
CONSTRAINTS;
```

Задания для самостоятельной работы.

1. Создайте таблицу DEPT из столбцов DEPARTMENT_ID и DEPARTMENT_NAME.
2. Заполните таблицу DEPT данными из таблицы DEPARTMENTS. Включите только нужные вам столбцы.
3. Создайте таблицу EMP на основе экземпляра таблицы EMPLOYEES.
4. Измените столбец в таблице EMP так, чтобы он вмещал фамилии большей длины. Проверьте изменение.
5. Создайте таблицу EMPLOYEES2 на основе структуры таблицы EMPLOYEES, включив только столбцы EMPLOYEE_ID, FIRST_NAME, LAST_NAME, SALARY и DEPARTMENT_ID. Присвойте столбцам новой таблицы имена ID, FIRST_NAME, LAST_NAME, SALARY и DEPT_ID.
6. Удалите таблицу EMP.
7. Переименуйте таблицу EMPLOYEES2 в EMP.
8. Удалите столбец FIRST_NAME из таблицы EMP. Проверьте описание таблицы, чтобы убедиться, что столбец удален.
9. Добавьте ограничение PRIMARY KEY на уровне таблицы EMP, используя столбец ID. Назовите ограничение my_emp_id_pk.
10. Создайте ограничение PRIMARY KEY для таблицы DEPT, используя столбец DEPARTMENT_ID. Ограничение должно получить название при создании. Назовите ограничение my_dept_id_pk.
11. Добавьте столбец DEPT_ID к таблице EMP. Добавьте ссылку в определение таблицы EMP, благодаря которой служащий не сможет быть приписан к несуществующему отделу. Назовите ограничение my_emp_dept_id_fk.
12. Измените таблицу EMP . Добавьте в нее столбец COMMISSION с типом данных NUMBER точностью 2 и масштабом 2. Добавьте ограничение для столбца COMMISSION , чтобы величина комиссионных была больше нуля.

Лабораторная работа №14. DDL. Определение других объектов БД.

Цель работы: Научиться создать и изменять: представления, индексы, последовательности.

Теоретическая часть.

Представление (view) – это сохраненный именованный SQL-запрос, к которому можно обращаться как к таблице. Представления нужны:

- Для ограничения доступа к базе данных.
- Для упрощения сложных запросов.
- Для обеспечения независимости от данных.
- Для представления одних и тех же данных в разных видах.

Различают простые и сложные представления:

Характеристика	Простые	Сложные
Количество таблиц	Одна	Одна или более
Содержат функции	Нет	Да
Содержат группы данных (предложение DISTINCT или групповые функции)	Нет	Да
Выполнение операций DML с представлениями	Да	Не всегда

Для создания представлений используется команда CREATE VIEW:

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW представление
[(псевдоним[, псевдоним]...)]
AS подзапрос
[WITH CHECK OPTION [CONSTRAINT ограничение]]
[WITH READ ONLY [CONSTRAINT ограничение]];
```

Например, данное представление содержит информацию о сотрудниках 80-го отдела:

```
CREATE VIEW empvu80
AS SELECT employee_id, last_name, salary
FROM employees
WHERE department_id = 80;

DESC empvu80
```

При создании представления можно (или нужно, в случае с функциями) использовать псевдонимы столбцов:

```
CREATE VIEW salvu50
AS SELECT employee_id ID_NUMBER, last_name NAME,
salary*12 ANN_SALARY
```

```
FROM employees
WHERE department_id = 50;

SELECT * FROM salvu50;
```

Изменить существующее представление можно при помощи команды CREATE OR REPLACE VIEW:

```
CREATE OR REPLACE VIEW empvu80
(id_number, name, sal, department_id)
AS SELECT employee_id, first_name || ' ' || last_name,
salary, department_id
FROM employees
WHERE department_id = 80;
Создание сложного представления:
CREATE VIEW dept_sum_vu
(name, minsal, maxsal, avgsal)
AS SELECT d.department_name, MIN(e.salary),
MAX(e.salary),AVG(e.salary)
FROM employees e, departments d
WHERE e.department_id = d.department_id
GROUP BY d.department_name;
```

Основное отличие простых представлений от сложных состоит в том, какие DML операции можно с ними осуществлять:

- Операции DML можно выполнять с простыми представлениями.
- Нельзя удалить строку, если представление содержит:
    - Групповые функции
    - Предложение GROUP BY
    - Ключевое слово DISTINCT
    - Ссылку на псевдостолбец ROWNUM.
- Невозможно изменить данные в представлении, которое содержит:
    - Групповые функции
    - Предложение GROUP BY
    - Ключевое слово DISTINCT
    - Ссылку на псевдостолбец ROWNUM
    - Столбцы, определенные с помощью выражений.
- Невозможно добавить данные в представление, которое содержит:
    - Групповые функции
    - Предложение GROUP BY
    - Ключевое слово DISTINCT
    - Ссылку на псевдостолбец ROWNUM
    - Столбцы, определенные с помощью выражений

- Невозможно добавить данные в представление, которое не содержит определенные в базовых таблицах столбцы с ограничением NOT NULL.

Для запрета любых DML операций можно создать представление с опцией WITH READ ONLY. Для контроля того, что при выполнении DML данные останутся в пределах выборки, определенной подзапросом, можно использовать опцию WITH CHECK OPTION:

```
-- Создаем простую view
CREATE OR REPLACE VIEW empvu10
(employee_number, employee_name, job_title)
AS SELECT employee_id, last_name, job_id
FROM employees
WHERE department_id = 10;

-- операция обновления разрешена
UPDATE empvu10 SET employee_name = employee_name||'.';

-- обратим внимание, что данные изменились в таблице
SELECT last_name FROM employees WHERE department_id = 10;
ROLLBACK;

--создадим view с опцией
CREATE OR REPLACE VIEW empvu10
(employee_number, employee_name, job_title)
AS SELECT employee_id, last_name, job_id
FROM employees
WHERE department_id = 10
WITH READ ONLY;

-- операция обновления теперь запрещена
UPDATE empvu10 SET employee_name = employee_name||'.';

-- создадим view с опцией WITH CHECK OPTION
CREATE OR REPLACE VIEW empvu10
(employee_number, employee_name, job_title, dep_id)
AS SELECT employee_id, last_name, job_id, department_id
FROM employees
WHERE department_id = 10
WITH CHECK OPTION CONSTRAINT empvu10_ck;

-- изменение не запрещено,
-- пока строки остаются в рамках выборки
UPDATE empvu10 SET employee_name = employee_name||'.';
UPDATE empvu10 SET dep_id = 20;
ROLLBACK;
```

Для удаления представления используется команда DROP VIEW:

```
DROP VIEW empvu10;
```

Рассмотрим еще два типа объектов БД: последовательности и индексы.

Последовательности нужны, как правило, для автоматической генерации значений первичного ключа. Последовательности – объекты совместного использования, гарантирующие уникальность генерируемых номеров, т.е. никакой элемент последовательности не будет выдан дважды. Для создания последовательности используется команда:

```
CREATE SEQUENCE последовательность
[INCREMENT BY n]
[START WITH n]
[{MAXVALUE n | NOMAXVALUE}]
[{MINVALUE n | NOMINVALUE}]
[{CYCLE | NOCYCLE}]
[{CACHE n | NOCACHE}];
```

Например:

```
CREATE SEQUENCE dept_deptid_seq
INCREMENT BY 10
START WITH 500
MAXVALUE 9999
NOCACHE
NOCYCLE;
```

У последовательности определены два псевдостолбца: CURRVAL – текущее значение последовательности и NEXTVAL – следующее значение последовательности. Использовать последовательности можно как в SELECT таки в DML операциях:

```
INSERT INTO departments(department_id,
department_name, location_id)
VALUES (dept_deptid_seq.NEXTVAL,
'Support', 2500);
SELECT dept_deptid_seq.CURRVAL FROM dual;
SELECT dept_deptid_seq.NEXTVAL FROM dual;
SELECT dept_deptid_seq.NEXTVAL FROM dual;
```

Изменить последовательность можно при помощи команды ALTER SEQUENCE, например:

```
ALTER SEQUENCE dept_deptid_seq
INCREMENT BY 20
MAXVALUE 999999
NOCACHE
```

```
NOCYCLE;
```

Для удаления последовательности используется команда DROP SEQUENCE:

```
DROP SEQUENCE dept_deptid_seq;
```

Для ускорения выборки данных сервер Oracle может использовать индексы. После создания индексы автоматически используются и поддерживаются сервером Oracle.

Индексы могут создаваться:

- Автоматически (при создании ограничений PRIMARY KEY или UNIQUE)
- Вручную пользователем для ускорения доступа к данным.

Индексы создаются по одному или нескольким столбцам, для этого используется команда CREATE INDEX:

```
CREATE INDEX индекс
ON таблица (столбец[, столбец]...);
```

Например:

```
CREATE INDEX emp_last_name_idx
ON employees(last_name);
```

Индекс следует создавать вручную, когда:

- Столбец имеет большой диапазон значений.
- Столбец содержит большое количество неопределенных значений.
- Один или более столбцов часто используются вместе в предложении WHERE или условии соединения.
- Таблица большая, и предполагается, что каждый запрос будет выбирать менее 2–4% строк.

Не следует создавать индекс, если:

- Таблица небольшого размера
- Столбцы редко используются как условие в запросе
- Предполагается, что каждый запрос будет выбирать более 2–4% строк
- Таблица часто обновляется.
- На столбцы, входящие в индекс, делаются ссылки в выражении

Текущее состояние индексов для таблицы можно выяснить следующим запросом:

```
SELECT ic.index_name, ic.column_name,
ic.column_position col_pos,ix.uniqueness
FROM user_indexes ix, user_ind_columns ic
WHERE ic.index_name = ix.index_name
```

```
AND ic.table_name = 'EMPLOYEES'
ORDER BY 1,3;
```

Обратим внимание, что запросы

```
SELECT * FROM employees WHERE last_name='King';
SELECT * FROM employees WHERE last_name like 'Ki%';
```

будут использовать построенный индекс, а запросы

```
SELECT * FROM employees WHERE upper(last_name)='KING';
SELECT * FROM employees WHERE last_name like '%ing';
```

использовать индекс не смогут и ускорение запроса не произойдет. В некоторых случаях, можно исправить ситуацию, построив индекс, основанный на функции:

```
CREATE INDEX upper_emp_name_idx
ON employees(UPPER(last_name));
```

Этот индекс сможет быть использован запросом

```
SELECT * FROM employees WHERE upper(last_name)='KING';
```

Для удаления индексов используется команда DROP INDEX:

```
DROP INDEX upper_emp_name_idx;
```

Задания для самостоятельной работы.

1. Создайте представление EMPLOYEES_VU. Включите номер служащего, фамилию служащего и номер отдела из таблицы EMPLOYEES. Смените заголовок столбца с фамилией служащего на EMPLOYEE.
2. Выведите содержимое представления EMPLOYEES_VU.
3. Из представления словаря данных USER_VIEWS выберите столбцы VIEW_NAME и TEXT.
4. Используя свое представление EMPLOYEES_VU , создайте запрос для вывода всех фамилий и номеров отделов служащих.
5. Создайте представление DEPT50, содержащее номер служащего, фамилию служащего и номер отдела для всех служащих отдела 50. Назовите столбцы представления EMPNO, EMPLOYEE и DEPTNO. Запретите операцию перевода служащего в другой отдел через представление (WITH CHECK OPTION).
6. Выведите структуру и содержимое представления DEPT50.
7. Попробуйте сменить номер отдела служащего по фамилии Matos на 80.
8. Создайте представление SALARY_VU , включающее фамилию служащего, название отдела, оклад и категорию оклада для всех служащих. Используйте таблицы

EMPLOYEES, DEPARTMENTS и JOB_GRADES. Соответственно, назовите столбцы Employee, Department, Salary и Grade.

9. Создайте последовательность для столбца главного ключа таблицы DEPT. Последовательность должна начинаться с 200 и иметь максимальное значение 1000. Шаг приращения значений – 10. Назовите последовательность DEPT_ID_SEQ.
10. Добавьте в таблицу два отдела: Education и Administration. Проверьте успешное выполнение вставки.
11. Создайте в таблице EMP неуникальный индекс для столбца таблицы DEPT_ID, имеющего ограничение FOREIGN KEY.
12. Выведите из словаря данных имена индексов для таблицы EMP и информацию об их уникальности.

Лабораторная работа №15. Хранимые процедуры/функции.

Цель работы: Научиться создать простейшие процедуры, функции и триггеры на языке PL/SQL.

Теоретическая часть.

Для написания хранимых процедур, функций и триггеров в Oracle используется язык программирования PL/SQL. В задачи данной работы не входит обучение даже основам этого языка, будут рассмотрены лишь некоторые примеры программ на этом языке, наглядно демонстрирующие разные области его применения.

Наиболее интересным с точки зрения использования SQL представляются хранимые функции и триггеры.

Хранимые функции PL/SQL могут быть использованы в SQL наравне со встроенными функциями, поэтому возможность разработки собственных функций, конечно, значительно расширяет возможности самого языка SQL, а также позволяет, в некоторых случаях, существенно улучшить производительность запросов и их читаемость. Например, если нам требуется вывести список сотрудников в виде "Имя Фамилия (Должность)" и дополнительно отметить знаком "*" тех, кто работает меньше 3х месяцев, то можно для этого написать SQL запрос, а можно создать для этого функцию, принимающую в качестве параметра ID сотрудника и возвращающую его имя в нужном нам формате. Реализовать эту функцию можно вот так:

```
CREATE OR REPLACE FUNCTION get_emp_record(pemp_id number)
RETURN varchar2
is
lret  varchar2(128);
ldate date;
BEGIN
  SELECT e.first_name||' '||e.last_name||'
('||j.job_title||')',e.hire_date
  INTO lret,ldate
  FROM employees e, jobs j
  WHERE e.job_id = j.job_id (+)
    AND e.employee_id = pemp_id;
  IF ldate > add_months(sysdate,-3) THEN
     lret:=lret||'*';
  END IF;
  RETURN lret;
END;
/
```

После чего ее можно использовать в запросах:

```
SELECT get_emp_record(employee_id) FROM employees;
```

В SQL предусмотрен специальный механизм триггеров, позволяющий обрабатывать различные события, происходящие внутри сервера баз данных. Триггер – это процедура на языке PL/SQL которая автоматически вызывается при возникновении определенного события. Чаще всего триггеры используются для дополнительного контроля и незначительной модификации данных при изменении эти данных в таблицах. Можно назначить исполнение триггера как до реального изменения данных в таблице (before insert/update/delete), так и после (after insert/update/delete). Можно даже определить триггер instead of, который будет выполняться вместо DML операции (используется для DML над представлениями). Общий синтаксис создания триггера:

```
CREATE [OR REPLACE] TRIGGER schema.trigger-name
{BEFORE | AFTER | INSTEAD OF} dml-event ON table-name
[FOR EACH ROW]
[DECLARE ...]
BEGIN
  -- PL/SQL code.
[EXCEPTION ...]
END;
/
```

Например, рассмотрим триггер, который перед вставкой новой записи в таблицу сотрудников:

- Заполняет столбец employee_id (если он не определен явно) следущим значением из последовательности EMPLOYEES_SEQ
- Приводит имя и фамилию к общему виду (первая буква прописная, остальные – строчные).

```
CREATE OR REPLACE TRIGGER bi_emp
BEFORE insert on employees
FOR EACH ROW
BEGIN
  IF :new.employee_id IS NULL THEN
     SELECT employees_seq.nextval INTO :new.employee_id FROM
dual;
  END IF;
  :new.first_name := initcap(:new.first_name);
  :new.last_name  := initcap(:new.last_name);
END;
/
```

Обратите внимание, что внутри тела триггера можно использовать две служебных переменных типа запись, в которых содержится старое (до момента начала исполнения DML) значение :old и новое (которое должно получится в результате DML) значение :new

для всех столбцов. В случае INSERT не определены старые значения, а в случае DELETE – новые. Значение записи :new можно изменять, при этом в таблицу попадут именно эти измененные данные.

В PL/SQL предусмотрен механизм порождения и обработки исключительных ситуаций. Если выполнение триггера прерывается возникновением исключительной ситуации, то прерывается и выполнение породившей его выполнение команды DML. Этим можно пользоваться для организации дополнительного контроля целостности данных, в случаях, когда на уровне стандартных ограничений этого сделать нельзя. Например, мы хотим запретить сотруднику быть начальником, если он проработал в текущей должности менее года. Для этого создадим триггер, который будет выполняться перед вставкой новой или изменением существующей записи в таблицу отделов:

```
CREATE OR REPLACE TRIGGER bi_dep
BEFORE insert or update on departments
FOR EACH ROW
DECLARE ldate date;
BEGIN
  IF :new.manager_id IS NOT NULL THEN
     SELECT hire_date INTO ldate FROM employees
      WHERE employee_id = :new.manager_id;
     IF add_months(ldate,12) > sysdate  THEN
        raise VALUE_ERROR;
     END IF;
  END IF;
END;
/
```

Проверим:

```
UPDATE departments SET manager_id = 100
 WHERE department_id = 260;

UPDATE employees SET hire_date = sysdate - 100
 WHERE employee_id = 200;

UPDATE departments SET manager_id = 200
 WHERE department_id = 260;

ROLLBACK;
```

Задания для самостоятельной работы.

1. Используя образец из теоретического раздела, напишите функцию, которая будет принимать в качестве параметров ID должности и сумму зарплаты, а возвращать 1,

если зарплата попадает в диапазон MIN_SALARY..MAX_SALARY из таблицы JOBS или 0 в противном случае.

2. Напишите триггер, использующий функцию из предыдущего задания, который будет отслеживать изменение зарплаты сотрудника. В случае выхода за границы диапазона триггер должен прервать выполнение операции с ошибкой VALUE_ERROR.