1. Основные понятия теории баз данных: база данных, система управления базами данных, основные требования к информации в БД, модели данных, логическая схема БД, основная терминология реляционных баз данных.

· **База данных (БД)**: это совокупность взаимосвязанных данных под управлением СУБД

(или Организованная коллекция данных, которая хранится и управляется таким образом, чтобы обеспечить легкий доступ, управление и обновление.)

· **Система управления базами данных (СУБД)**: Программное обеспечение, которое используется для создания, управления и манипулирования базами данных. Примеры включают MySQL, PostgreSQL, Oracle и Microsoft SQL Server.

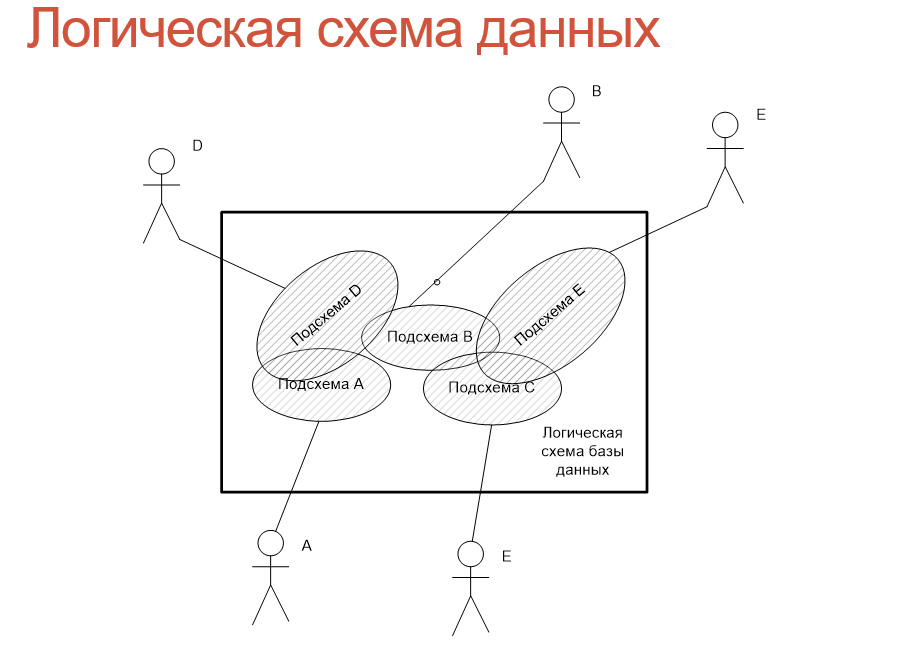
· **Основные требования к информации в БД**:

* **Полезность: уменьшает информационную энтропию системы**
* **Полнота информации: информации должно быть достаточно, чтобы осуществить качественное управление**
* **Точность:**
* **Достоверность: заведомо ошибочные данные не должны хранится в базе данных**
* **Непротиворечивость:**
* **АктуальностьЖ**

· **Модели данных**: Структуры, которые определяют, как данные организованы и как они связаны друг с другом. Основные модели включают:

* **Иерархическая модель**: Данные организованы в виде дерева.
* **Сетевая модель**: Данные организованы в виде графа.
* **Реляционная модель**: Данные организованы в виде таблиц.

· **Логическая схема БД**: Абстрактное представление структуры базы данных, которое описывает, как данные организованы и как они связаны друг с другом.



Совокупность всех представлений - **это логическая схема данных**

· **Основная терминология реляционных баз данных**:

* **Таблица**: Основная структура данных в реляционной базе данных, состоящая из строк и столбцов.
* **Строка (запись)**: Один экземпляр данных в таблице.
* **Столбец (поле)**: Атрибут или характеристика данных в таблице.
* **Ключ**: Уникальный идентификатор записи в таблице. Примеры включают первичный ключ и внешний ключ.
* **Запрос**: Операция, выполняемая для извлечения данных из базы данных.

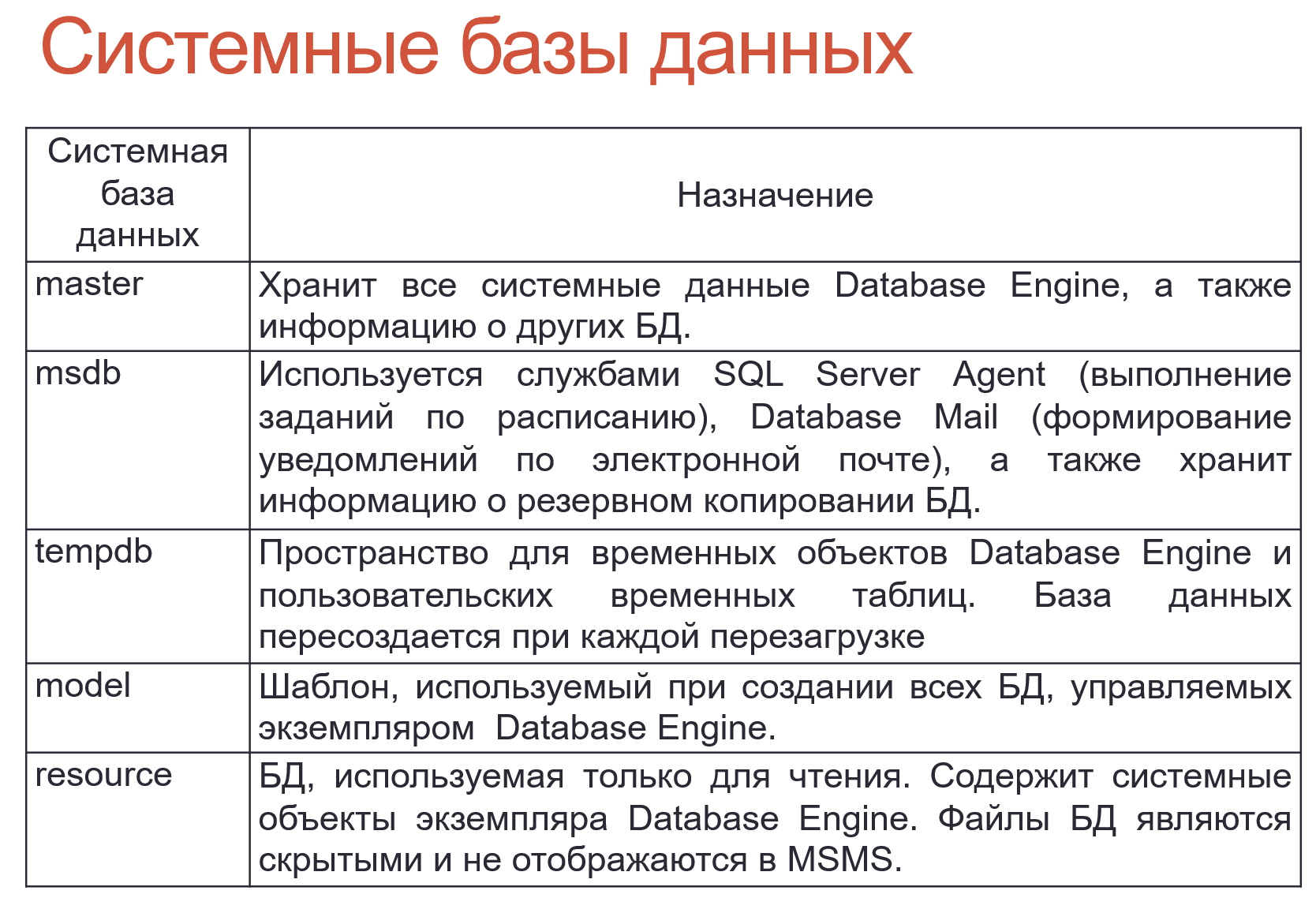
1. Язык SQL.

SQL (Structured Query Language, язык структурированных запросов) — специализированный язык, предназначенный для написания запросов к реляционной БД.

* 1986 – первый вариант стандарта
* 1989 – доработан стандарт
* 1992 – внесены значительные изменения (SQL2)
* 1999 – (SQL3) добавлены:
  + поддержка регулярных выражений
  + рекурсивных запросов
  + поддержка триггеров
  + базовые процедурные расширения
  + нескалярные типы данных
  + некоторые объектно-ориентированные возможности
* 2003 - поддержка XML
* 2006 - возможность совместно использовать в запросах SQL и XQuery
* 2008 - улучшены возможности оконных функций

Вот основные аспекты SQL:

1. **DDL (Data Definition Language)**: Операторы DDL предназначены для **создания**, **удаления** и **изменения** объектов БД или сервера СУБД.
   * CREATE: Создание новых таблиц или баз данных.
   * ALTER: Изменение структуры существующих таблиц.
   * DROP: Удаление таблиц или баз данных.
2. **DML (Data Manipulation Language)**: Команды, используемые для манипулирования данными в базе данных.
   * SELECT: Извлечение данных из таблиц.
   * INSERT: Вставка новых данных в таблицы.
   * UPDATE: Обновление существующих данных.
   * DELETE: Удаление данных из таблиц.
3. **DCL (Data Control Language)**: Команды, используемые для управления доступом к данным.
   * GRANT: Предоставление прав пользователям.
   * REVOKE: Отзыв прав у пользователей.
   * DENY: ? в лекции
4. **TCL (Transaction Control Language)**: Команды, используемые для управления транзакциями.(Транзакция – это несколько DML-операторов, которые либо **все** выполняются, либо все не выполняются )
   * COMMIT: Фиксация изменений в базе данных.
   * ROLLBACK: Отмена изменений, сделанных в транзакции.
   * SAVEPOINT: Создание точки сохранения внутри транзакции.
5. **Основные операторы и функции**:
   * **Операторы сравнения**: =, <>, >, <, >=, <=.
   * **Логические операторы**: AND, OR, NOT.
   * **Агрегатные функции**: COUNT, SUM, AVG, MAX, MIN.
6. Системные базы данных в SQL Server: master, msdb, model, tempdb.



* **Database Engine** является ядром системы управления реляционной БД

1. Структура файла базы данных в SQL Server. Файловые группы.

БД представляет собой набор файлов операционной системы трех типов: первичный файл (расширение **mdf**), вторичные файлы (**ndf**) и файлы журнала транзакций (**log**). Все файлы БД, кроме файлов журнала транзакций, распределены по файловым группам.

Файловые группы – это поименованный набор файлов БД.

* Все файлы БД, кроме файлов журнала транзакций, распределены по файловым группам
  + Первичные
  + Вторичные
* Используются для упрощения администрирования
  + Секционирование
  + Резервное копирование и восстановление
* Во вторичных файловых группах могут располагаться только вторичные файлы.
* В первичной файловой группе помимо обязательного первичного файла тоже могут быть расположены вторичные файлы.
* При создании таблиц и индексов дисковая память для них автоматически отводится в файловой группе по умолчанию.
* Для размещения в другой файловой группе следует явно указывать ее имя в операторе CREATE, создающем таблицу или индекс.

Copilot:

В SQL Server структура файла базы данных и файловые группы играют важную роль в управлении данными. Вот основные аспекты:

1. **Файлы базы данных**:
   * **Основной файл данных (Primary Data File)**: Это основной файл базы данных, который содержит начальные данные и информацию о структуре базы данных. Обычно имеет расширение .mdf.
   * **Вторичные файлы данных (Secondary Data Files)**: Дополнительные файлы данных, которые могут использоваться для распределения данных по нескольким дискам. Обычно имеют расширение .ndf.
   * **Файлы журнала транзакций (Transaction Log Files)**: Эти файлы содержат информацию о всех изменениях, сделанных в базе данных, и используются для восстановления данных в случае сбоя. Обычно имеют расширение .ldf.
2. **Файловые группы**:
   * **Основная файловая группа (Primary Filegroup)**: Содержит основной файл данных и любые вторичные файлы данных, которые не были явно назначены другим файловым группам.
   * **Пользовательские файловые группы (User-defined Filegroups)**: Пользователь может создавать дополнительные файловые группы для распределения данных по разным дискам и улучшения производительности. Эти файловые группы могут содержать вторичные файлы данных.

Файловые группы позволяют более гибко управлять данными и оптимизировать производительность базы данных, распределяя данные по разным физическим дискам. Это особенно полезно для больших баз данных с высоким уровнем нагрузки.

1. Нормализация таблиц базы данных. Нормальные формы таблиц.

Нормализация — это процесс, в результате выполнения которого таблицы базы данных проверяются на наличие зависимостей между столбцами и, если необходимо, то исходная таблица разделяется на несколько таблиц.

Основные нормальные формы таблиц включают:

1. **Первая нормальная форма (1NF)**: Таблица находится в 1NF, если все ее атрибуты (значения полей) содержат только атомарные (неделимые и не вычисляемые) значения, и каждая запись в них уникальна.
2. **Вторая нормальная форма (2NF)**: Таблица находится во 2NF, если она находится в 1NF и все неключевые поля полностью зависят от первичного ключа.
3. **Третья нормальная форма (3NF)**: Таблица находится в 3NF, если она находится во 2NF и все неключевые поля в таблицах не зависят взаимно друг от друга.
4. **Бойс-Кодд нормальная форма (BCNF)**: Таблица находится в BCNF, если она находится в 3NF и каждый детерминант является кандидатом на ключ.
5. **Четвертая нормальная форма (4NF)**: Таблица находится в 4NF, если она находится в BCNF и не содержит многозначных зависимостей.
6. **Пятая нормальная форма (5NF)**: Таблица находится в 5NF, если она находится в 4NF и не содержит соединительных зависимостей.

Первичный ключ — в один из потенциальных ключей отношения, выбранный в качестве основного ключа

* Внешний ключ ̶̶ подмножество атрибутов некоторой переменной отношения R2, значения которых должны совпадать со значениями некоторого потенциального ключа некоторой переменной отношения R1.
* Соблюдение ссылочной целостности

1. Таблицы и их типы данных Microsoft SQL Server. - !!!!!-доработать-!!!!!

В Microsoft SQL Server таблицы являются основными структурами для хранения данных. Каждая таблица состоит из строк и столбцов, где строки представляют записи, а столбцы — атрибуты данных. Вот основные типы данных, которые можно использовать в таблицах SQL Server:

1. **Числовые типы данных**:
   * INT: Целые числа от -2,147,483,648 до 2,147,483,647.
   * BIGINT: Целые числа от -9,223,372,036,854,775,808 до 9,223,372,036,854,775,807.
   * SMALLINT: Целые числа от -32,768 до 32,767.
   * TINYINT: Целые числа от 0 до 255.
   * DECIMAL и NUMERIC: Числа с фиксированной точностью и масштабом.
   * FLOAT и REAL: Числа с плавающей точкой.
2. **Символьные типы данных**:
   * CHAR: Фиксированная длина строк.
   * VARCHAR: Переменная длина строк.
   * TEXT: Большие текстовые данные.
3. **Юникодные символьные типы данных**:
   * NCHAR: Фиксированная длина строк с поддержкой Юникода.
   * NVARCHAR: Переменная длина строк с поддержкой Юникода.
   * NTEXT: Большие текстовые данные с поддержкой Юникода.
4. **Дата и время**:
   * DATE: Дата (год, месяц, день).
   * TIME: Время (часы, минуты, секунды).
   * DATETIME: Дата и время.
   * SMALLDATETIME: Дата и время с меньшей точностью.
   * DATETIME2: Дата и время с большей точностью.
   * DATETIMEOFFSET: Дата и время с часовым поясом.
5. **Двоичные типы данных**:
   * BINARY: Фиксированная длина двоичных данных.
   * VARBINARY: Переменная длина двоичных данных.
   * IMAGE: Большие двоичные данные.
6. **Другие типы данных**:
   * BIT: Логическое значение (0 или 1).
   * MONEY и SMALLMONEY: Денежные значения.
   * UNIQUEIDENTIFIER: Уникальный идентификатор (GUID).
   * XML: Данные в формате XML.
   * JSON: Данные в формате JSON (начиная с SQL Server 2016).

| **Тип данных** | **Описание** | **Тип данных** | **Описание** |
| --- | --- | --- | --- |
| **integer**  **(int)** | Целочисленные значения, занимают 4 байта. | **date** | Дата, занимает 3 байта |
| **smallint** | Целочисленные значения, занимают 2 байта | **time[(p)]**  **0 ≤ p ≤ 7** | Время, занимает от 3 до 5 байтов. **p** − колич. знаков после точки в секундах |
| **tinyint** | Неотрицательные целочисленные значения, занимают 1 байт | **smalldatetime** | Дата и время, занимает 2 байта |
| **bigint** | Целочисленные значения, занимают 8 байт | **datetime** | Дата и время, занимает 4 байта |
| **real** | Вещественные числа с плавающей точкой | **datetime2** | Дата и время, занимает 8 байтов |
| **decimal (p[,s] )**  **(dec)**  или  **numeric (p[,s] )** | Вещественные значения с фикс. точкой, **р** − общее количество цифр, **s** − количество цифр после точки. Занимает от 5 до 17 байт. | **сhаr[(n)]** | Строки фиксированной длины из однобайтовых симво­лов, n − количество символов |
| **float[(p)]** | Вещественные числа с плавающей точкой. Если **р** < 25 − одинарная точность (4 байта), при **р** > 25 − двойная точность (8 байтов) | **nchar[(n)]** | Строки фиксированной длины символов Unicode. Каждый символ занимает 2 байта. |
| **money** | Денежные значения, занимают 8 байтов | **varchar[(n)]** | Строки переменной длины из однобайтовых симво­лов |
| **smallmoney** | Денежные значения, занимают 4 байта | **nvarchar[(n)]** | Строки переменной длины символов Unicode − 2 байта |
| **binary(n)** | Задает битовую строку, длиной ровно **n** байтов | **varbinary(n)** | Задает битовую строку, длиной не более **n** байтов |

1. Подзапросы. Конструкции in, exists, all, any, some.

**Подзапрос** – это SELECT-запрос, который выполняется в рамках другого запроса. Подзапросы могут применяться в секции WHERE. Подзапросы бывают двух видов: **коррелируемые и независимые.**

**Коррелируемый подзапрос** зависит от внешнего запроса и выполняется для каждой строки результирующего набора.

**Независимый подзапрос** не зависит от внешнего запроса и выполняется только один раз, но результат его выполнения подставляется в каждую строку результирующего набора.

Может применяться со следующими операторами:

* оператором IN;
* операторами сравнения и EXISTS;
* операторами ANY и ALL;

1. **IN**: Операция IN формирует логическое значение «истина» в том случае, если значение, указанное слева от ключевого слова IN, равно хотя бы одному из значений списка, указанного справа.

SELECT \* FROM employees WHERE department\_id IN (SELECT department\_id FROM departments WHERE location = 'New York');

1. **EXISTS**: Операция EXISTS формирует значение «истина», если результирующий набор подзапроса содержит хотя бы одну строку, в противоположном случае − значение «ложь».

SELECT \* FROM employees WHERE EXISTS (SELECT 1 FROM departments WHERE departments.department\_id = employees.department\_id AND location = 'New York');

1. **ALL**: Операция >= ALL формирует истинное значение в том случае, когда значение стоящее слева больше или равно каждому значению в списке, указанном справа.

SELECT \* FROM employees WHERE salary > ALL (SELECT salary FROM employees WHERE department\_id = 10);

1. **ANY** (или SOME): Операция >= ANY формирует истинное значение в том случае, если значение стоящее слева, больше или равно хотя бы одному значению в списке, указанном справа.

SELECT \* FROM employees WHERE salary > ANY (SELECT salary FROM employees

1. Группировка данных с использованием cube, rollup

Основное назначение **группировки** с помощью секции GROUP BY – разбиение множества строк, сформированных секциями FROM и WHERE, на группы в соответствии со значениями в заданных столбцах, а также выполнение вычислений над группами строк с помощью наиболее часто используемых функций: **AVG** (вычисление среднего значения), **COUNT** (вычисление количества строк), **MAX** (вычисление максимального значения), MIN (вычисление минимального значения), **SUM** (вычисление суммы значений).

При использовании секции **GROUP BY** в SELECT-списке допускается указывать только те столбцы, по которым осуществляется группировка.

**Аналитическими** запросами к базе данных принято называть SELECT-запросы, сводные (агрегатные) результаты которых вычисляются над данными, хранящимися в таблицах базы данных. К ним относятся запросы, содержащие конструкции **ROLLUP** и **CUBE**. Они используются в секции GROUP BY и служат для вычисления значений агрегатных функций для подмножеств строк.

· **GROUP BY**: Используется для группировки строк по одному или нескольким столбцам и выполнения агрегатных функций на каждой группе.

SELECT department\_id, COUNT(\*)

FROM employees

GROUP BY department\_id;

· **GROUP BY HAVING**: Используется для фильтрации групп, созданных оператором GROUP BY, на основе условия.

SELECT department\_id, COUNT(\*)

FROM employees

GROUP BY department\_id

HAVING COUNT(\*) > 10;

· **GROUP BY WITH CUBE**: Создает все возможные комбинации группировок, включая промежуточные итоги и общий итог.

SELECT department\_id, job\_id, SUM(salary)

FROM employees

GROUP BY CUBE (department\_id, job\_id);

**Правило формирования** результирующего набора SELECT-запроса, применяющего конструкцию **CUBE**:

1. **Формируется множество всех подмножеств выражений, указанных в CUBE-списке**: Это означает, что для каждого столбца, указанного в списке CUBE, создаются все возможные комбинации группировок. Например, если у вас есть столбцы A, B и C, то будут созданы подмножества: {A}, {B}, {C}, {A, B}, {A, C}, {B, C}, {A, B, C}.
2. **Для каждого непустого подмножества выполняется группировка**: Это означает, что для каждого подмножества столбцов будет выполнена группировка данных. Например, для подмножества {A, B} будет выполнена группировка по столбцам A и B.
3. **Если количество элементов подмножества меньше количества элементов CUBE-списка, то соответствующие значения заполняются значением NULL**: Это означает, что если подмножество не включает все столбцы из списка CUBE, то столбцы, которые не включены в подмножество, будут заполнены значением NULL. Например, если подмножество {A, B}, то столбец C будет заполнен значением NULL.
4. **Для пустого подмножества выполняется группировка, аналогичная той, что выполнялась для ROLLUP**: Это означает, что будет создана одна строка, которая представляет общий итог по всем данным, где все столбцы будут заполнены значениями NULL.

Пример:

sql

SELECT department\_id, job\_id, SUM(salary)

FROM employees

GROUP BY CUBE (department\_id, job\_id);

Если у вас есть столбцы department\_id и job\_id, результирующий набор будет содержать:

1. Группировку по department\_id и job\_id.
2. Группировку по department\_id (столбец job\_id будет NULL).
3. Группировку по job\_id (столбец department\_id будет NULL).
4. Общий итог (оба столбца будут NULL).

· **GROUP BY WITH ROLLUP**: Создает иерархические группы, добавляя промежуточные итоги и общий итог.

**Правило формирования** результирующего набора SELECT-запроса, применяющего конструкцию **ROLLUP**:

1. **Результирующий набор содержит n + 1 групп строк**, где n – количество выражений для группировки столбцов, указанных за словом ROLLUP. Это означает, что если у вас есть 3 столбца для группировки, результирующий набор будет содержать 4 группы строк.
2. **Первая группа строк** является результатом выполнения группировки по всем n выражениям. Это стандартная группировка по всем указанным столбцам.
3. **Вторая группа строк** является результатом группировки первой группы строк по n - 1 первым выражениям. Столбцы, по которым не выполнялась группировка, заполняются значениями NULL. Например, если у вас есть столбцы A, B и C, то вторая группа будет сгруппирована по столбцам A и B, а столбец C будет заполнен значением NULL.
4. **Группа строк k** является группировкой группы строк, полученной на предыдущем этапе, по n – k + 1 первым выражениям. Это означает, что каждая следующая группа будет группироваться по одному столбцу меньше, чем предыдущая.
5. **Последняя (n + 1)-я группа** содержит одну строку. Это общий итог по всем данным, где все столбцы, по которым выполнялась группировка, будут заполнены значениями NULL.

Пример:

sql

SELECT department\_id, job\_id, SUM(salary)

FROM employees

GROUP BY ROLLUP (department\_id, job\_id);

Если у вас есть столбцы department\_id и job\_id, результирующий набор будет содержать:

1. Группировку по department\_id и job\_id.
2. Группировку по department\_id (столбец job\_id будет NULL).
3. Общий итог (оба столбца будут NULL).
4. Операторы union (all), intersect, except.

Оператор **UNION** выполняет теоретико-множественную операцию объединения результирующих наборов SELECT-запросов, в котором строки не могут повторяться.

Если требуется механическое объединение строк, можно применить оператор **UNION ALL**.

Результатом оператора **INTERSECT** является набор строк, являющийся пересечением двух исходных результирующих наборов SELECT-запросов.

Результатом оператора **EXCEPT** является набор строк, являющийся разностью двух исходных результирующих наборов SELECT-запросов (т.е. в результат включаются те строки, которые есть в первом запросе, но отсутствуют во втором).

· **UNION**: Объединяет результаты двух или более SELECT-запросов, удаляя дубликаты. Все запросы должны иметь одинаковое количество столбцов с совместимыми типами данных.

SELECT column1, column2 FROM table1

UNION

SELECT column1, column2 FROM table2;

· **UNION ALL**: Объединяет результаты двух или более SELECT-запросов, включая дубликаты. Это быстрее, чем UNION, так как не выполняется удаление дубликатов.

SELECT column1, column2 FROM table1

UNION ALL

SELECT column1, column2 FROM table2;

· **INTERSECT**: Возвращает общие строки из двух SELECT-запросов. Все запросы должны иметь одинаковое количество столбцов с совместимыми типами данных.

SELECT column1, column2 FROM table1

INTERSECT

SELECT column1, column2 FROM table2;

· **EXCEPT**: Возвращает строки из первого SELECT-запроса, которых нет во втором SELECT-запросе. Все запросы должны иметь одинаковое количество столбцов с совместимыми типами данных.

SELECT column1, column2 FROM table1

EXCEPT

SELECT column1, column2 FROM table2;

1. Операторы соединения таблиц.

Операторы соединения таблиц (JOIN) в SQL Server позволяют объединять строки из двух или более таблиц на основе связанных столбцов. Вот основные типы соединений:

1. **INNER JOIN**: Возвращает только те строки, которые имеют совпадающие значения в обеих таблицах.

SELECT employees.name, departments.department\_name

FROM employees

INNER JOIN departments ON employees.department\_id = departments.department\_id;

1. **LEFT JOIN (или LEFT OUTER JOIN)**: Возвращает все строки из левой таблицы и совпадающие строки из правой таблицы. Если совпадений нет, то столбцы из правой таблицы будут заполнены значениями NULL.

SELECT employees.name, departments.department\_name

FROM employees

LEFT JOIN departments ON employees.department\_id = departments.department\_id;

1. **RIGHT JOIN (или RIGHT OUTER JOIN)**: Возвращает все строки из правой таблицы и совпадающие строки из левой таблицы. Если совпадений нет, то столбцы из левой таблицы будут заполнены значениями NULL.

SELECT employees.name, departments.department\_name

FROM employees

RIGHT JOIN departments ON employees.department\_id = departments.department\_id;

1. **FULL JOIN (или FULL OUTER JOIN)**: Возвращает все строки, когда есть совпадение в одной из таблиц. Если совпадений нет, то столбцы из соответствующей таблицы будут заполнены значениями NULL.

SELECT employees.name, departments.department\_name

FROM employees

FULL JOIN departments ON employees.department\_id = departments.department\_id;

1. **CROSS JOIN**: Возвращает декартово произведение двух таблиц, то есть все возможные комбинации строк из обеих таблиц (соединяет каждую строку первой таблицы с каждой строкой второй таблицы).

SELECT employees.name, departments.department\_name

FROM employees

CROSS JOIN departments;

1. **SELF JOIN**: Соединение таблицы с самой собой. Используется для сравнения строк внутри одной таблицы.

SELECT e1.name AS Employee1, e2.name AS Employee2

FROM employees e1

INNER JOIN employees e2 ON e1.manager\_id = e2.employee\_id;

1. Язык T-SQL. Пакеты. Объявление переменных. Операторы присвоения Оператор цикла while.

· **Язык T-SQL (Transact-SQL)**: Это расширение языка SQL, разработанное Microsoft для работы с SQL Server. T-SQL включает дополнительные функции, такие как управление потоком выполнения, обработка ошибок и работа с переменными.

· **Пакеты**: Пакет в T-SQL — это набор команд, которые выполняются как единое целое. Пакеты могут содержать объявления переменных, операторы присвоения, циклы и другие конструкции.

· **Объявление переменных**: Переменные в T-SQL объявляются с использованием ключевого слова DECLARE. Пример:

DECLARE @variable\_name datatype;

· **Операторы присвоения**: Для присвоения значения переменной используется оператор SET или SELECT. Примеры:

SET @variable\_name = value;

SELECT @variable\_name = value;

· **Оператор цикла WHILE**: Оператор WHILE используется для выполнения блока команд, пока условие истинно. Пример:

DECLARE @counter INT = 0;

WHILE @counter < 10

BEGIN

PRINT @counter;

SET @counter = @counter + 1;

END;

1. Язык T-SQL. Операторы print, if-else, case. Операторы begin-end, waitfor и return.

· **Операторы вывода данных**: Вывод данных в T-SQL возможен двумя способами: оператором SELECT можно сформировать выходной результирующий набор и с помощью оператора PRINT можно вывести строку в стандартный выходной поток. Примеры:

PRINT ‘name = ’ + cast(@variable\_name as varchar(10));

SELECT @variable\_name = value;

· MSS поддерживает широкий набор встроенных математических функций, которые можно применять в сценариях T-SQL для вычислений:

* **print 'Округление: ' + cast(round(12345.12345, 2) as varchar(12));**
* **print 'Нижнее целое: ' + cast(floor(24.5) as varchar(12));**
* **print 'Возведение в степень: '+ cast(power(12.0, 2) as varchar(12));**
* **print 'Логарифм : ' + cast(log(144.0) as varchar(12));**
* **print 'Корень квадратный: ' + cast(sqrt(144.0) as varchar(12));**
* **print 'Экпонента: ' + cast(exp(4.96981) as varchar(12));**
* **print 'Абсолютное значение: '+ cast(abs(-5) as varchar(12));**
* **print 'Cинус: ' + cast(sin(pi()) as varchar(12));**
* **print 'Подстрока: ' + substring('1234567890', 3,2);**
* **print 'Удалить пробелы справа: ' + rtrim('12345 ') +'X';**
* **print 'Удалить пробелы слева : ' + 'X'+ ltrim(' 67890');**
* **print 'Нижний регистр: ' + lower ('ВЕРХНИЙ РЕГИСТР');**
* **print 'Верхний регистр: ' + upper ('нижний регистр');**
* **print 'Заменить: ' + replace('1234512345', '5', 'X');**
* **print 'Строка пробелов: ' + 'X'+ space(5) +'X';**
* **print 'Повторить строку: ' + replicate('12', 5);**
* **print 'Найти по шаблону: ' + cast (patindex ('%Y\_Y%', '123456YxY7890') as varchar(5));**

· **IF-ELSE**: Используется для выполнения условных операций.

IF @value > 10

BEGIN

PRINT 'Value is greater than 10';

END

ELSE

BEGIN

PRINT 'Value is 10 or less';

END;

· **CASE**: Используется для выполнения условных операций внутри выражений.

SELECT

CASE

WHEN @value > 10 THEN 'Greater than 10'

WHEN @value = 10 THEN 'Equal to 10'

ELSE 'Less than 10'

END AS ValueDescription;

· **BEGIN-END**: Используется для группировки нескольких операторов в один блок.

BEGIN

PRINT 'Start of block';

-- другие операторы

PRINT 'End of block';

END;

· **WAITFOR**: Используется для задержки выполнения на определенное время или до определенного времени.

WAITFOR DELAY '00:01:00'; -- задержка на 1 минуту

WAITFOR TIME '12:00:00'; -- задержка до 12:00

· **RETURN**: Используется для завершения выполнения процедуры и возврата значения.

RETURN @return\_value;

1. Язык T-SQL. Обработка ошибок в конструкциях try-catch. Функция RAISERROR.

· **TRY-CATCH**: Конструкция TRY-CATCH используется для обработки ошибок в T-SQL. Код внутри блока TRY выполняется, и если возникает ошибка, управление передается в блок CATCH, где можно обработать ошибку.

BEGIN TRY

-- Код, который может вызвать ошибку

SELECT 1 / 0; -- Пример ошибки деления на ноль

END TRY

BEGIN CATCH

-- Обработка ошибки

PRINT 'Произошла ошибка: ' + ERROR\_MESSAGE();

END CATCH;

· **Функция RAISEERROR**: Функция RAISEERROR используется для генерации пользовательских ошибок и передачи управления в блок CATCH. Она позволяет задать сообщение об ошибке, уровень серьезности и состояние ошибки.

BEGIN TRY

-- Код, который может вызвать ошибку

IF @value < 0

BEGIN

RAISEERROR('Значение не может быть отрицательным', 16, 1);

END;

END TRY

BEGIN CATCH

-- Обработка ошибки

PRINT 'Произошла ошибка: ' + ERROR\_MESSAGE();

END CATCH;

1. Локальные и глобальные временные таблицы в SQL Server.

Основное отличие временных таблиц от постоянных в том, что они хранятся в системной базе данных TEMPDB и удаляются после окончания сеанса работы с базой.

Локальные временные таблицы имеют имена, начинающиеся с символа # и доступны только пользователю, ее создавшему.

Глобальные временные таблицы имеют имена, начинающиеся с символов ## и доступны всем пользователям, подключенным к серверу.

В SQL Server существуют два типа временных таблиц: локальные и глобальные. Вот их основные характеристики:

1. **Локальные временные таблицы**:
   * Создаются с использованием префикса # перед именем таблицы.
   * Доступны только в пределах текущей сессии или соединения.
   * Автоматически удаляются, когда сессия или соединение закрывается.
   * Пример создания локальной временной таблицы:

CREATE TABLE #TempTable (

ID INT,

Name NVARCHAR(50)

);

1. **Глобальные временные таблицы**:
   * Создаются с использованием префикса ## перед именем таблицы.
   * Доступны для всех сессий и соединений на сервере.
   * Автоматически удаляются, когда все сессии, использующие эту таблицу, закрываются.
   * Пример создания глобальной временной таблицы:

CREATE TABLE ##GlobalTempTable (

ID INT,

Name NVARCHAR(50)

);

1. Курсоры в SQL Server. Объявление курсора. Общая схема работы с курсором: declare, open, fetch, close, deallocate. Типы курсоров.

Курсор является программной конструкцией, которая дает возможность пользователю обрабатывать строки результирующего набора запись за записью. Курсоры бывают локальные и глобальные (по умолчанию), статические и динамические (по умолчанию).

1. **Курсоры**: Курсоры позволяют выполнять построчную обработку набора данных, возвращаемого запросом. Они полезны, когда необходимо выполнить операции над каждой строкой результата запроса.
2. **Объявление курсора**: Курсор объявляется с использованием оператора DECLARE.

DECLARE cursor\_name CURSOR FOR

SELECT column1, column2 FROM table\_name;

1. **Общая схема работы с курсором**:
   * **DECLARE**: Объявление курсора.

DECLARE cursor\_name CURSOR FOR

SELECT column1, column2 FROM table\_name;

* + **OPEN**: Открытие курсора для выполнения запроса и получения набора данных.

OPEN cursor\_name;

* + **FETCH**: Извлечение следующей строки из набора данных курсора. Результат каждого считывания проверяется с помощью системной функции @@FETCH\_STATUS, которая возвращает значение 0, если оператор FETCH выполнен успешно; −1, если достигнут конец результирующего набора и строка не считывается; −2, если выбранная строка отсутствует в БД.

FETCH NEXT FROM cursor\_name INTO @variable1, @variable2;

* + **CLOSE**: Закрытие курсора после завершения работы с ним.

CLOSE cursor\_name;

* + **DEALLOCATE**: Освобождение ресурсов, связанных с курсором. Если курсор глобальный, то он должен быть освобожден с использованием оператора DEALLOCATE.

DEALLOCATE cursor\_name;

1. **Типы курсоров**:
   * **LOCAL**: Локальный курсор может применяться в рамках одного пакета и ресурсы, выделенные ему при объявлении, освобождаются сразу после завершения работы пакета.

**DECLARE Tovary CURSOR LOCAL**

**for SELECT Наименование, Цена from Товары;**

**DECLARE @tv char(20), @cena real;**

**OPEN Tovary;**

**fetch Tovary into @tv, @cena;**

**print '1. '+@tv+cast(@cena as varchar(6));**

**go**

**DECLARE @tv char(20), @cena real;**

**fetch Tovary into @tv, @cena;**

**print '2. '+@tv+cast(@cena as varchar(6));**

**go**

* + **GLOBAL:** Глобальный курсор может быть объявлен, открыт и использован в разных пакетах. Выделенные ему при объявлении ресурсы освобождаются только после выполнения оператора DEALLOCATE или при завершении сеанса пользователя.

**DECLARE Tovary CURSOR GLOBAL**

**for SELECT Наименование, Цена from Товары;**

**DECLARE @tv char(20), @cena real;**

**OPEN Tovary;**

**fetch Tovary into @tv, @cena;**

**print '1. '+@tv+cast(@cena as varchar(6));**

**go**

**DECLARE @tv char(20), @cena real;**

**fetch Tovary into @tv, @cena;**

**print '2. '+@tv+cast(@cena as varchar(6));**

**go**

* + **STATIC**: Открытие статического курсора приводит к выгрузке результирующего набора во временную таблицу системной БД **TEMPDB**, и все дальнейшие операции осуществляются с этой таблицей. После открытия курсора все текущие изменения в исходных таблицах не будут отражаться в результирующем наборе.

**DECLARE @tid char(10), @tnm char(40), @tgn char(1);**

**DECLARE Zakaz CURSOR LOCAL STATIC**

**for SELECT Наименование\_товара, Цена\_продажи, Количество**

**FROM dbo.Заказы where Заказчик = 'Луч';**

**open Zakaz;**

**print 'Количество строк : '+cast(@@CURSOR\_ROWS as varchar(5));**

**UPDATE Заказы set Количество = 5 where Наименование\_товара = 'Стул';**

**DELETE Заказы where Наименование\_товара = 'Шкаф';**

**INSERT Заказы (Номер\_заказа, Наименование\_товара, Цена\_продажи,**

**Количество, Дата\_поставки, Заказчик)**

**values (18, 'Шкаф', 340, 1, '2014-08-02', 'Луч');**

**FETCH Zakaz into @tid, @tnm, @tgn;**

**while @@fetch\_status = 0**

**begin**

**print @tid + ' '+ @tnm + ' '+ @tgn;**

**fetch Zakaz into @tid, @tnm, @tgn;**

**end;**

**CLOSE Zakaz;**

Здесь значение функции @@CURSOR\_ROWS равно -**n** (количество записей) при асинхронной выборке, равно **n** при синхронной выборке, равно **0**, если курсор не открыт. При выполнении курсора все изменения (UPDATE, DELETE и INSERT) в исходной таблице **Заказы** не отражаются на результате выборки строк.

Если заменить LOCAL STATICна LOCAL DYNAMIC, то изменения будут отражаться в результирующем наборе.

* + **DYNAMIC**: Отражает все изменения, внесенные в базу данных после открытия курсора.

1. По умолчанию для курсора установлен атрибут SCROLL, позволяющий применять оператор FETCH с дополнительными опциями позиционирования.
2. В примере ниже выводятся названия товаров, которые заказаны фирмой с названием «Луч». Перед каждым названием выводится номер строки результирующего набора, определяемый функцией ROW\_NUMBER(). Выбор строки определяется соответствующим ключевым словом в операторе FETCH.

**DECLARE @tc int, @rn char(50);**

**DECLARE Primer1 cursor local dynamic SCROLL**

**for SELECT row\_number() over (order by Наименование\_товара) N,**

**Наименование\_товара FROM dbo.Заказы**

**where Заказчик = 'Луч'**

**OPEN Primer1;**

**FETCH Primer1 into @tc, @rn;**

**print 'следующая строка : ' + cast(@tc as varchar(3))+ rtrim(@rn);**

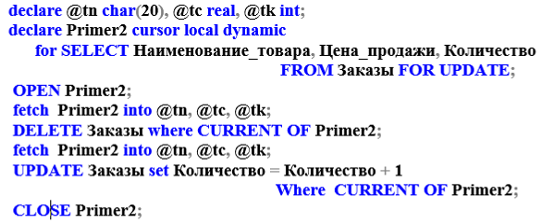
**FETCH LAST from Primer1 into @tc, @rn;**

**print 'последняя строка : ' + cast(@tc as varchar(3))+ rtrim(@rn);**

**. . . . . . . . . . . . . . . . . . . . . . . . .**

**CLOSE Primer1;**

Курсоры с установленным свойством FOR UPDATE помимо чтения данных из строк с помощью оператора FETCH, могут эти строки изменять или удалять с помощью операторов UPDATE и DELETE, если в секции WHERE эти операторы используют операцию CURRENT OF, для которой указывается имя курсора (в примере ниже Primer2). Такой формат операторов позволяет удалять или изменять строки в таблице (в примере таблица **Заказы**), соответствующих *текущей* позиции курсора в результирующем наборе.



1. Хранимые процедуры и функции T-SQL. Создание хранимых процедур. Передача параметров. Входные и выходные параметры.

· **Хранимые процедуры**: Это предварительно скомпилированные наборы T-SQL команд (блоков BEGIN…END), которые хранятся в базе данных и могут быть вызваны по имени. Они позволяют выполнять повторяющиеся задачи и улучшать производительность.

Также:

* Принимает входные параметры
* Принимаете и формирует выходные параметры
* Целочисленное значение, возвращаемое к точке вызова с помощью оператора RETURN
* Один или более результирующих наборов, сформированных операторами SELECT
* Содержимое стандартного выходного потока, полученного при выполнении операторов PRINT
* Результирующий набор хранимой процедуры может быть использован в качестве источника строк для INSERT

Допускается применение:

* Основных DDL, DML и TCL-операторов
* Конструкций TRY/CATCH
* Курсоров
* Временных таблиц

Не допускается применение:

* CREATE or ALTER FUNCTION
* CREATE or ALTER TRIGGER
* CREATE or ALTER PROCEDURE
* CREATE or ALTER VIEW
* USE databasename

· **Создание хранимых процедур**: Хранимая процедура создается с использованием оператора CREATE PROCEDURE.

CREATE PROCEDURE ProcedureName

AS

BEGIN

-- T-SQL команды

SELECT \* FROM table\_name;

END;

· **Передача параметров**: Хранимые процедуры могут принимать входные и выходные параметры.

* **Входные параметры**: Передаются в процедуру при вызове.

CREATE PROCEDURE ProcedureName

@Parameter1 INT,

@Parameter2 NVARCHAR(50)

AS

BEGIN

-- Использование параметров

SELECT \* FROM table\_name WHERE column1 = @Parameter1 AND column2 = @Parameter2;

END;

* **Выходные параметры**: Возвращают значения из процедуры.

CREATE PROCEDURE ProcedureName

@InputParameter INT,

@OutputParameter INT OUTPUT

AS

BEGIN

-- Присвоение значения выходному параметру

SET @OutputParameter = @InputParameter \* 2;

END;

· **Вызов хранимой процедуры**: Хранимая процедура вызывается с использованием оператора EXEC.

DECLARE @Result INT;

EXEC ProcedureName @InputParameter = 5, @OutputParameter = @Result OUTPUT;

PRINT @Result; -- Вывод значения выходного параметра

· **Функции**: Функции в T-SQL возвращают значение и могут использоваться в выражениях. Они создаются с использованием оператора CREATE FUNCTION.

CREATE FUNCTION FunctionName (@Parameter1 INT)

RETURNS INT

AS

BEGIN

RETURN @Parameter1 \* 2;

END;

· **Вызов функции**: Функция вызывается в выражениях.

SELECT dbo.FunctionName(5);

1. Хранимые процедуры и функции T-SQL. Виды функций. Создание функций. Передача параметров.
2. **Хранимые процедуры**: Это предварительно скомпилированные наборы T-SQL команд, которые хранятся на сервере и могут быть вызваны по имени. Они позволяют выполнять повторяющиеся задачи и улучшать производительность.
3. **Функции**: Функции в T-SQL возвращают значение и могут использоваться в выражениях. Они создаются с использованием оператора CREATE FUNCTION.

**Виды функций:**

* **Скалярные функции**: Возвращают одно значение (нельзя timestamp, cursor, text, ntext, image, table). Пример:

CREATE FUNCTION FunctionName (@Parameter1 INT)

RETURNS INT

AS

BEGIN

RETURN @Parameter1 \* 2;

END;

* **Табличные (Inline) функции**: Возвращают таблицу (одна команда SELECT). Пример:

CREATE FUNCTION TableFunctionName (@Parameter1 INT) RETURNS TABLE

AS

RETURN SELECT Column1, Column2

FROM SomeTable

WHERE Column1 = @Parameter1;

* **Многооператорные табличные (Multi-Statement) функции**: Возвращают таблицу и могут содержать несколько операторов. Пример:

CREATE FUNCTION MultiStatementTableFunction (@Parameter1 INT)

RETURNS @ResultTable TABLE (Column1 INT, Column2 NVARCHAR(50))

AS

BEGIN

-- Несколько операторов

INSERT INTO @ResultTable

SELECT Column1, Column2 FROM SomeTable WHERE Column1 = @Parameter1;

INSERT INTO @ResultTable

SELECT Column1, Column2 FROM AnotherTable WHERE Column1 = @Parameter1;

RETURN;

END;

**Передача параметров:**

* **Входные параметры**: Передаются в функцию при вызове.

CREATE FUNCTION FunctionName (@Parameter1 INT)

RETURNS INT

AS

BEGIN

RETURN @Parameter1 \* 2;

END;

* **Выходные параметры**: Возвращают значения из функции. Пример для хранимой процедуры:

CREATE PROCEDURE ProcedureName

@InputParameter INT,

@OutputParameter INT OUTPUT

AS

BEGIN

-- Присвоение значения выходному параметру

SET @OutputParameter = @InputParameter \* 2;

END;

**Вызов функции:**

Функция вызывается в выражениях.

SELECT dbo.FunctionName(5);

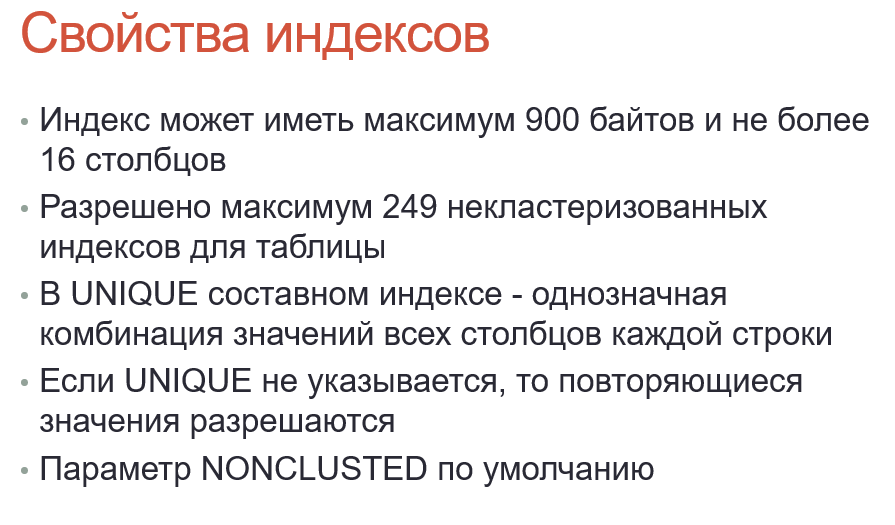
1. Индексы. Назначение и применение индексов. Виды индексов. Применение различных видов индексов. Оптимизация запросов.

Индекс – это объект базы данных, позволяющий ускорить поиск в определенной таблице, так как при этом данные организуются в виде сбалансированного бинарного дерева поиска.

* Индекс представляет собой отдельную физическую структуру данных, которая позволяет получать быстрый доступ к одной или нескольким строкам данных
* Индексы сохраняются в страницах индексов
* Для каждой индексируемой строки имеется элемент индекса, который сохраняется на странице индексов
* Каждый элемент индекса состоит из ключа индекса и указателя
* Индексы создаются по сбалансированному дереву B+
* B+-дерево имеет древовидную структуру, в которой все листья находятся на расстоянии одинакового количества уровней от вершины дерева
* Это свойство поддерживается при добавлении или удалении данных в индексированном столбце
* Индекс всегда связан с таблицей или с подмножеством столбцов таблицы

Как и любой другой объект базы данных, индекс может быть создан с помощью оператора CREATE, модифицирован с помощью ALTER и удален с помощью оператора DROP. Для одной таблицы возможно построение нескольких индексов.

Индексы бывают кластеризованные, некластеризованные, уникальные, неуникальные и др.



· **Назначение и применение индексов**:

* Индексы используются для ускорения выполнения запросов, обеспечивая быстрый доступ к данным в таблицах.
* Они создаются на одном или нескольких столбцах таблицы и позволяют быстро находить строки, соответствующие условиям запроса.

· **Виды индексов**:

* **Кластерный индекс (Clustered Index)**: Определяет физический порядок строк в таблице. В таблице может быть только один кластерный индекс. Обычно кластеризованные индексы создаются автоматически при создании таблицы если в ней присутствует первичный ключ (ограничение PRIMARY KEY). Кластеризованные индексы физически упорядочены в соответствии со значениями индексируемых столбцов. В таблице может быть только один кластеризованный индекс.
  + Определяет физический порядок данных в таблице
  + Может только один для одной таблицы
  + Таблица перестраивается в порядке индекса
  + Листья дерева индекса содержат страницы данных
  + Создается по умолчанию для каждой таблицы, для которой определен первичный ключ
  + Уникальный – в столбце, для которого определен кластеризованный индекс, каждое значение данных может встречаться только один раз
  + Если кластеризованный индекс создается для столбца, содержащего повторяющиеся значения, СУБД принудительно добавляет четырехбайтовый идентификатор к строкам, содержащим дубликаты значений

CREATE CLUSTERED INDEX #idx\_name ON #table\_name (column\_name asc);

* **Некластерный индекс (Non-Clustered Index)**: Некластеризованные индексы не влияют на физический порядок строк в таблице. Создает отдельную структуру, которая указывает на физическое расположение данных. В таблице может быть несколько некластерных индексов.
  + физически находится отдельно от таблицы
  + страницы листьев состоят из ключей индекса и закладок
  + может быть несколько для одной таблицы
  + не изменяет физическое упорядочивание строк таблицы
  + Если есть кластеризованный индекс, то закладка некластеризованного индекса показывает B+-дерево кластеризованного индекса таблицы
  + Если нет кластеризованного индекса, закладка идентична RID — Row Identifier, состоящего из:
  + Адреса файла, в котором хранится таблица,
  + Адреса физического блока (страницы), в котором хранится строка,
  + Смещения строки в странице.
  + Поиск данных с использованием некластеризованного индекса в зависимости от типа таблицы:
  + Куча — прохождение при поиске по структуре некластеризованного индекса, после чего строка извлекается, используя идентификатор строки.
  + Кластеризованная таблица — прохождение при поиске по структуре некластеризованного индекса, после чего следует прохождение по соответствующему кластеризованному индексу.

CREATE NONCLUSTERED INDEX idx\_name ON table\_name (column\_name);

* **Уникальный индекс (Unique Index)**: Гарантирует, что значения в индексируемом столбце или столбцах будут уникальными.

CREATE UNIQUE INDEX idx\_name ON table\_name (column\_name);

* **Фильтрованный индекс (Filtered Index)**: Создается на подмножестве строк таблицы, что позволяет уменьшить размер индекса и улучшить производительность.
  + Фильтрующий индекс создается только для строк таблицы, которые удовлетворяют логическому условию
  + Если множество строк, выбираемое WHERE-выражением SELECT-запроса, является подмножеством множества индексируемых фильтрующим индексом строк, оптимизатор будет его применять в плане запроса
  + Можно объединять с индексами покрытия

CREATE NONCLUSTERED INDEX idx\_name ON table\_name (column\_name) WHERE condition;

* **Полнотекстовый индекс (Full-Text Index)**: Используется для полнотекстового поиска в текстовых столбцах.

CREATE FULLTEXT INDEX ON table\_name (column\_name) KEY INDEX idx\_name;

· **Применение различных видов индексов**:

* **Кластерные индексы**: Используются для столбцов, по которым часто выполняются сортировка и диапазонные запросы.
* **Некластерные индексы**: Используются для столбцов, по которым часто выполняются точечные запросы и фильтрация.
* **Уникальные индексы**: Используются для обеспечения уникальности значений в столбцах.
* **Фильтрованные индексы**: Используются для столбцов с большим количеством повторяющихся значений и для улучшения производительности запросов с условиями.
* **Полнотекстовые индексы**: Используются для поиска по текстовым данным, таким как документы и статьи.

· **Оптимизация запросов**:

* Использование индексов позволяет значительно ускорить выполнение запросов, уменьшая количество операций ввода-вывода и повышая производительность.
* Важно правильно выбирать столбцы для создания индексов и учитывать типы запросов, которые будут выполняться.
* Регулярное обновление статистики и анализ производительности запросов помогают поддерживать оптимальную работу базы данных.

1. План запроса. Этапы обработки select запроса. Понятие стоимости запроса. Понятия селективности и плотности. Индексы. Реорганизация, перестроение, включение и отключение индексов. !!!!!!-Copilot-!!!!!!

### План запроса

План запроса — это набор шагов, которые SQL Server выполняет для выполнения запроса. Он включает в себя информацию о том, как данные будут извлечены и обработаны.

### Этапы обработки SELECT запроса

1. **FROM**: Определение источника данных.
2. **WHERE**: Фильтрация строк на основе условий.
3. **GROUP BY**: Группировка строк.
4. **HAVING**: Фильтрация групп.
5. **SELECT**: Выбор столбцов.
6. **ORDER BY**: Сортировка результатов.
7. **TOP**: Ограничение количества возвращаемых строк.

### Понятие стоимости запроса

Стоимость запроса — это оценка ресурсов, необходимых для выполнения запроса, таких как процессорное время и операции ввода-вывода. SQL Server использует эту оценку для выбора оптимального плана выполнения запроса.

### Понятия селективности и плотности

* **Селективность**: Мера уникальности значений в столбце. Чем выше селективность, тем меньше дубликатов и тем эффективнее индекс.
* **Плотность**: Обратная величина селективности. Чем выше плотность, тем больше дубликатов и тем менее эффективен индекс.

### Индексы

Индексы используются для ускорения выполнения запросов, обеспечивая быстрый доступ к данным.

### Реорганизация, перестроение, включение и отключение индексов

* **Реорганизация**: Процесс дефрагментации индекса без его полного перестроения. Выполняется с помощью команды ALTER INDEX ... REORGANIZE.
* **Перестроение**: Полное перестроение индекса для устранения фрагментации. Выполняется с помощью команды ALTER INDEX ... REBUILD.
* **Включение индекса**: Включение ранее отключенного индекса. Выполняется с помощью команды ALTER INDEX ... REBUILD.
* **Отключение индекса**: Отключение индекса для временного исключения его из использования. Выполняется с помощью команды ALTER INDEX ... DISABLE.

1. Триггеры. Типы триггеров. Создание after-триггера. !!!!-Copilot-!!!!

**Триггеры**

Триггеры — это специальные хранимые процедуры, которые автоматически выполняются при возникновении определенных событий в таблице, таких как вставка, обновление или удаление данных.

**Типы триггеров**

1. **AFTER (или FOR) триггеры**: Выполняются после выполнения операции INSERT, UPDATE или DELETE. Они используются для выполнения действий после изменения данных.
2. **INSTEAD OF триггеры**: Выполняются вместо выполнения операции INSERT, UPDATE или DELETE. Они используются для замены стандартного поведения операций.

**Создание AFTER-триггера**

Пример создания AFTER-триггера, который выполняется после вставки данных в таблицу:

CREATE TRIGGER trgAfterInsert

ON table\_name

AFTER INSERT

AS

BEGIN

-- Действия, выполняемые после вставки данных

INSERT INTO audit\_table (column1, column2)

SELECT column1, column2 FROM inserted;

END;

В этом примере триггер trgAfterInsert выполняется после вставки данных в таблицу table\_name. Он копирует вставленные данные в таблицу audit\_table для аудита.

1. Триггеры. Создание и назначение instead of-триггеров. Таблицы inserted, deleted. !!!!-Copilot-!!!!

**Триггеры**

Триггеры — это специальные хранимые процедуры, которые автоматически выполняются при возникновении определенных событий в таблице, таких как вставка, обновление или удаление данных.

**INSTEAD OF триггеры**

INSTEAD OF триггеры выполняются вместо выполнения стандартных операций INSERT, UPDATE или DELETE. Они позволяют переопределить поведение этих операций.

**Создание INSTEAD OF триггера**

Пример создания INSTEAD OF триггера для операции INSERT:

CREATE TRIGGER trgInsteadOfInsert

ON table\_name

INSTEAD OF INSERT

AS

BEGIN

-- Действия, выполняемые вместо вставки данных

INSERT INTO another\_table (column1, column2)

SELECT column1, column2 FROM inserted;

END;

В этом примере триггер trgInsteadOfInsert выполняется вместо вставки данных в таблицу table\_name. Вместо этого данные вставляются в таблицу another\_table.

**Таблицы inserted и deleted**

Таблицы inserted и deleted являются виртуальными таблицами, которые используются в триггерах для доступа к новым и старым значениям строк.

* **Таблица** inserted: Содержит новые строки, которые были вставлены или обновлены.
* **Таблица** deleted: Содержит старые строки, которые были удалены или обновлены.

Пример использования таблиц inserted и deleted в триггере:

CREATE TRIGGER trgInsteadOfUpdate

ON table\_name

INSTEAD OF UPDATE

AS

BEGIN

-- Действия, выполняемые вместо обновления данных

UPDATE another\_table

SET column1 = i.column1, column2 = i.column2

FROM inserted i

JOIN another\_table a ON i.id = a.id;

DELETE FROM another\_table

WHERE id IN (SELECT id FROM deleted);

END;

В этом примере триггер trgInsteadOfUpdate выполняется вместо обновления данных в таблице table\_name. Он обновляет данные в таблице another\_table и удаляет старые строки.

1. Транзакции. Свойства ACID. Транзакции. Уровни изолированности транзакций. Функция trancount.

**Транзакции**

Транзакция — это последовательность операций, выполняемых как единое целое. Если одна из операций не удается, все изменения, сделанные в рамках транзакции, откатываются, чтобы сохранить целостность данных.

Неявная транзакция — задает любую отдельную инструкцию INSERT, UPDATE или DELETE как единицу транзакции

Явная транзакция — группа инструкций, начало и конец которой обозначаются инструкциями:

* BEGIN TRANSACTION
* COMMIT
* ROLLBACK

**Свойства ACID**

ACID — это набор свойств, которые обеспечивают надежность транзакций:

1. **Atomicity (Атомарность)**: Транзакция выполняется полностью или не выполняется вовсе.
2. **Consistency (Согласованность)**: Транзакция переводит базу данных из одного согласованного состояния в другое.
3. **Isolation (Изолированность)**: Все параллельные транзакции отделяются друг от друга (Активная транзакция не может видеть модификации данных в параллельной или незавершенной транзакции).
4. **Durability (Долговечность)**: Транзакцию после фиксациинельзя отменить, кроме как другой транзакцией.

**Уровни изолированности транзакций**

Уровни изолированности определяют, как транзакции взаимодействуют друг с другом:

1. **Read Uncommitted**:
   * Не изолирует операции чтения других транзакций
   * Транзакция не задает и не признает блокировок

Допускает проблемы:

* + Грязное чтение
  + Неповторяемое чтение
  + Фантомное чтение

1. **Read Committed**:
   * Транзакция выполняет проверку только на наличие монопольной блокировки для данной строки
   * Является уровнем изоляции по умолчанию

Проблемы:

* + Неповторяемое чтение
  + Фантомное чтение

1. **Repeatable Read**:
   * Устанавливает разделяемые блокировки на все считываемые данные и удерживает эти блокировки до тех пор, пока транзакция не будет подтверждена или отменена
   * Не препятствует другим инструкциям вставлять новые строки

Проблема:

* + Фантомное чтение

1. **Serializable**:
   * Устанавливает блокировку на всю область данных, считываемых соответствующей транзакцией
   * Предотвращает вставку новых строк другой транзакцией до тех пор, пока первая транзакция не будет подтверждена или отменена
   * Реализуется с использованием метода блокировки диапазона ключа
   * Блокировка диапазона ключа блокирует элементы индексов

**Функция @@TRANCOUNT**

Функция @@TRANCOUNT возвращает количество активных транзакций для текущей сессии. Она полезна для определения, находится ли код внутри транзакции, и для управления вложенными транзакциями.

Пример использования транзакций и функции @@TRANCOUNT:

BEGIN TRANSACTION;

PRINT 'Количество активных транзакций: ' + CAST(@@TRANCOUNT AS NVARCHAR);

-- Выполнение операций

INSERT INTO table\_name (column1, column2) VALUES (value1, value2);

-- Проверка условий и фиксация или откат транзакции

IF @@ERROR = 0

BEGIN

COMMIT TRANSACTION;

PRINT 'Транзакция успешно завершена';

END

ELSE

BEGIN

ROLLBACK TRANSACTION;

PRINT 'Транзакция откатена';

END;

PRINT 'Количество активных транзакций: ' + CAST(@@TRANCOUNT AS NVARCHAR);

1. Операторы TCL. Привилегии. Роли. Назначение привилегий. !!!!-Copilot-!!!!

**Операторы TCL (Transaction Control Language)**

Операторы TCL используются для управления транзакциями в базе данных:

1. **BEGIN TRANSACTION**: Начинает новую транзакцию.

BEGIN TRANSACTION;

1. **COMMIT**: Фиксирует все изменения, сделанные в рамках текущей транзакции.

COMMIT;

1. **ROLLBACK**: Отменяет все изменения, сделанные в рамках текущей транзакции.

ROLLBACK;

1. **SAVEPOINT**: Создает точку сохранения внутри транзакции, к которой можно откатиться.

SAVE TRANSACTION savepoint\_name;

1. **ROLLBACK TO SAVEPOINT**: Откатывает транзакцию к указанной точке сохранения.

ROLLBACK TRANSACTION savepoint\_name;

**Привилегии**

Привилегии — это права доступа, которые предоставляются пользователям для выполнения определенных действий в базе данных. Примеры привилегий:

* **SELECT**: Право на чтение данных из таблицы.
* **INSERT**: Право на вставку данных в таблицу.
* **UPDATE**: Право на обновление данных в таблице.
* **DELETE**: Право на удаление данных из таблицы.
* **EXECUTE**: Право на выполнение хранимых процедур и функций.

**Роли**

Роли — это наборы привилегий, которые можно назначать пользователям для упрощения управления доступом. Примеры ролей:

* **db\_owner**: Полный доступ ко всем объектам базы данных.
* **db\_datareader**: Право на чтение всех данных в базе данных.
* **db\_datawriter**: Право на вставку, обновление и удаление данных в базе данных.
* **db\_ddladmin**: Право на выполнение команд DDL (Data Definition Language), таких как создание и изменение таблиц.

**Назначение привилегий**

Привилегии можно назначать пользователям и ролям с помощью операторов GRANT и REVOKE:

* **GRANT**: Предоставляет привилегии пользователю или роли.

GRANT SELECT ON table\_name TO user\_name;

* **REVOKE**: Отзывает привилегии у пользователя или роли.

REVOKE SELECT ON table\_name FROM user\_name;

1. Дистрибутивы СУБД Oracle. Установка СУБД Oracle 12с на Windows. Global Database Name и SID. !!!!-Copilot-!!!!

**Дистрибутивы СУБД Oracle**

Oracle Database 12c доступен для различных операционных систем, включая Windows, Linux и Unix. Дистрибутивы можно скачать с официального сайта Oracle после регистрации и принятия лицензионного соглашения.

**Установка СУБД Oracle 12c на Windows**

1. **Скачивание дистрибутива**: Скачайте установочные файлы Oracle Database 12c с официального сайта Oracle. Файлы обычно предоставляются в формате ZIP.
2. **Распаковка файлов**: Распакуйте скачанные ZIP-файлы в одну папку.
3. **Запуск установки**: Запустите файл setup.exe из распакованной папки, чтобы начать процесс установки.
4. **Следование инструкциям мастера установки**:
   * Введите адрес электронной почты для получения обновлений безопасности (можно пропустить).
   * Выберите опцию "Создать и настроить базу данных".
   * Выберите класс системы (обычно "Desktop class" для настольных систем).
   * Укажите учетную запись Windows для установки и настройки Oracle Home.
   * Укажите папку для установки, глобальное имя базы данных и пароль.
   * Проверьте информацию и нажмите кнопку "Установить".
5. **Завершение установки**: После завершения установки настройте пароли для учетных записей SYS и SYSTEM.

**Global Database Name и SID**

* **Global Database Name**: Глобальное имя базы данных состоит из имени базы данных (DB\_NAME) и домена базы данных (DB\_DOMAIN). Оно используется для уникальной идентификации базы данных в распределенной среде.

global\_name = DB\_NAME.DB\_DOMAIN

* **SID (System Identifier)**: Идентификатор системы (SID) — это уникальное имя экземпляра базы данных Oracle. Оно используется для идентификации конкретного экземпляра базы данных на сервере.

ORACLE\_SID = DB\_NAME

1. Основные системные пользователи. Основные специальные привилегии. Роль DBA. !!!!-Copilot-!!!!-не понимаю

### Основные системные пользователи

### SYS – предопределенный привилегированный пользователь ранга администратора базы данных, который является владельцем ключевых ресурсов БД Oracle

### SYSTEM – предопределенный привилегированный пользователь, которому принадлежат ключевые ресурсы БД Oracle

### Основные специальные привилегии

1. **CONTROL**: Полный доступ ко всем объектам и функциям базы данных или сервера.
2. **ALTER**: Право изменять объекты базы данных, такие как таблицы, представления и процедуры.
3. **CREATE**: Право создавать новые объекты базы данных, такие как таблицы, представления и процедуры.
4. **DROP**: Право удалять объекты базы данных.
5. **EXECUTE**: Право выполнять хранимые процедуры и функции.
6. **SELECT**: Право читать данные из таблиц и представлений.
7. **INSERT**: Право вставлять данные в таблицы.
8. **UPDATE**: Право обновлять данные в таблицах.
9. **DELETE**: Право удалять данные из таблиц.

### Роль DBA (Database Administrator)

Роль DBA включает в себя управление и поддержку баз данных. Основные обязанности DBA включают:

1. **Установка и настройка**: Установка и настройка SQL Server и баз данных.
2. **Мониторинг и производительность**: Мониторинг производительности баз данных и оптимизация запросов.
3. **Резервное копирование и восстановление**: Настройка и управление резервным копированием и восстановлением данных.
4. **Безопасность**: Управление доступом пользователей и ролями, обеспечение безопасности данных.
5. **Обслуживание и обновление**: Выполнение регулярного обслуживания баз данных и обновление программного обеспечения.
6. **Техническая поддержка**: Предоставление технической поддержки пользователям и разработчикам.
7. Понятия базы данных и экземпляра базы данных.

### База данных

База данных — это организованная коллекция данных, которая хранится и управляется с помощью системы управления базами данных (СУБД). Базы данных используются для хранения, управления и извлечения данных в различных приложениях и системах. Примеры баз данных включают реляционные базы данных, такие как SQL Server, Oracle и MySQL, а также нереляционные базы данных, такие как MongoDB и Cassandra.

### Экземпляр базы данных

Экземпляр базы данных — это конкретный экземпляр СУБД, который управляет одной или несколькими базами данных. Экземпляр включает в себя процессы, память и другие ресурсы, необходимые для управления базами данных. В SQL Server экземпляр базы данных может содержать несколько баз данных, каждая из которых имеет свои собственные схемы, таблицы и данные.

Пример:

* **База данных**: AdventureWorks, Northwind
* **Экземпляр базы данных**: SQL Server Instance1, который управляет базами данных AdventureWorks и Northwind

Экземпляр базы данных обеспечивает изоляцию и управление ресурсами для баз данных, что позволяет эффективно управлять и масштабировать системы баз данных.

1. Запуск и останов экземпляра базы данных Oracle.

**Запуск**

1. **STARTUP NOMOUNT**: Запуск экземпляра Oracle без монтирования базы данных. Доступ пользователей запрещен. Используется для создания БД и пересоздания управляющих файлов;
2. **STARTUP MOUNT:** Запустить экземпляр, монтировать БД, но не запускать ее. Считан файл параметров. Происходит обращение к управляющим файлам. Проверяется состояние файлов БД. Доступ пользователей запрещен. Из состояния STARTUP NOMOUNT переводится ALTER DATABASE MOUNT;
3. **STARTUP OPEN:** Запустить экземпляр. Монтировать и открыть БД. Могут присоединяться пользователи. Из состояния STARTUP MOUNT переводится ALTER DATABASE OPEN.

**Остановка**

1. **SHUTDOWN NORMAL:** Запрещено создавать новые сессии. Ожидается завершение работы всех пользователей. Самый безопасный и долгий способ останова. Никаких восстановительных работ при следующем старте не проводится;
2. **SHUTDOWN TRANSACTIONAL:** Запрещено создавать новые сессии. Запрещено запускать новые транзакции. Сервер дожидается завершения уже начатых транзакций и отключает пользователей, не имеющих активных транзакций. Применяется в случаях, когда нет возможности применить NORMAL. Никаких восстановительных работ при следующем старте не проводится.
3. **SHUTDOWN IMMEDIATE:** Запрещено создавать новые сессии. Запрещено запускать новые транзакции. Все незафиксированные транзакции откатываются. Применяется в случаях, когда нет возможности ждать. Никаких восстановительных работ при следующем старте не проводится.
4. **SHUTDOWN ABORT:** Применяется в крайних случаях, когда остальные режимы останова не приводят к результату. Все действия прекращаются. Все транзакции не фиксируются и не откатываются. Пользователей отсоединяют от БД. При следующем старте будет выполнено возможное восстановление.
5. Словарь базы данных: назначение, применение, основные представления.

**Словарь Oracle** - набор таблиц и связанных с ними представлений, который представляет возможность отследить внутреннюю структуру базы данных и деятельность СУБД Oracle

Создается при генерации базы данных.

Обновляется и обслуживается сервером Oracle в фоновом режиме после выполнения операторов DDL.

Позволяет запрашивать данные в виде представлений.

Содержит следующую информацию:

- Имена пользователей сервера Oracle

- Уровни привилегий пользователей

- Имена объектов базы данных

- Табличные ограничения

- Учетные данные

Располагается в системном табличном пространстве SYSTEM

Владелец: пользователь SYS, некоторые представления - SYSTEM

Для доступа к словарю необходима специальная привилегия GRANT SELECT ANY DICTIONARY

Словарь базы данных (Data Dictionary) — это набор системных таблиц и представлений, которые содержат метаданные о структуре базы данных, таких как таблицы, столбцы, индексы, ограничения и другие объекты базы данных. Словарь базы данных автоматически создается и поддерживается системой управления базами данных (СУБД).

**Назначение словаря базы данных**

1. **Хранение метаданных**: Словарь базы данных хранит информацию о структуре базы данных и ее объектах.
2. **Поддержка целостности данных**: Словарь базы данных обеспечивает целостность и согласованность данных, храня информацию о связях между объектами.
3. **Оптимизация запросов**: Словарь базы данных используется оптимизатором запросов для выбора наилучшего плана выполнения запросов.
4. **Администрирование базы данных**: Словарь базы данных предоставляет информацию, необходимую для управления и администрирования базы данных.

**Применение словаря базы данных**

1. **Разработка и проектирование**: Разработчики используют словарь базы данных для понимания структуры базы данных и ее объектов.
2. **Оптимизация производительности**: Администраторы базы данных используют словарь для анализа производительности запросов и оптимизации структуры базы данных.
3. **Обеспечение безопасности**: Словарь базы данных содержит информацию о пользователях, ролях и привилегиях, что помогает обеспечивать безопасность данных.
4. **Мониторинг и аудит**: Словарь базы данных используется для мониторинга изменений в структуре базы данных и проведения аудита.

**Основные представления словаря базы данных**

**USER**: Объекты, принадлежащие пользователю

**ALL**: Объекты, к которым пользователь имеет доступ

**DBA**: Все объекты базы данных (для администратора БД)

V$: Производительность сервера

Прочие представления (DICTIONARY и пр.)

1. **INFORMATION\_SCHEMA**: Стандартный набор представлений, предоставляющий информацию о таблицах, столбцах, индексах и других объектах базы данных.

SELECT \* FROM INFORMATION\_SCHEMA.TABLES;

SELECT \* FROM INFORMATION\_SCHEMA.COLUMNS;

1. **SYS**: Набор системных представлений, предоставляющих подробную информацию о структуре базы данных и ее объектах.

SELECT \* FROM sys.tables;

SELECT \* FROM sys.columns;

1. **DBA\_**: Представления, доступные администраторам базы данных, предоставляющие информацию о всех объектах базы данных.

SELECT \* FROM DBA\_TABLES;

SELECT \* FROM DBA\_TAB\_COLUMNS;

1. **ALL\_**: Представления, предоставляющие информацию о всех объектах базы данных, доступных текущему пользователю.

SELECT \* FROM ALL\_TABLES;

SELECT \* FROM ALL\_TAB\_COLUMNS;

1. **USER\_**: Представления, предоставляющие информацию о всех объектах базы данных, принадлежащих текущему пользователю.

SELECT \* FROM USER\_TABLES;

SELECT \* FROM USER\_TAB\_COLUMNS;

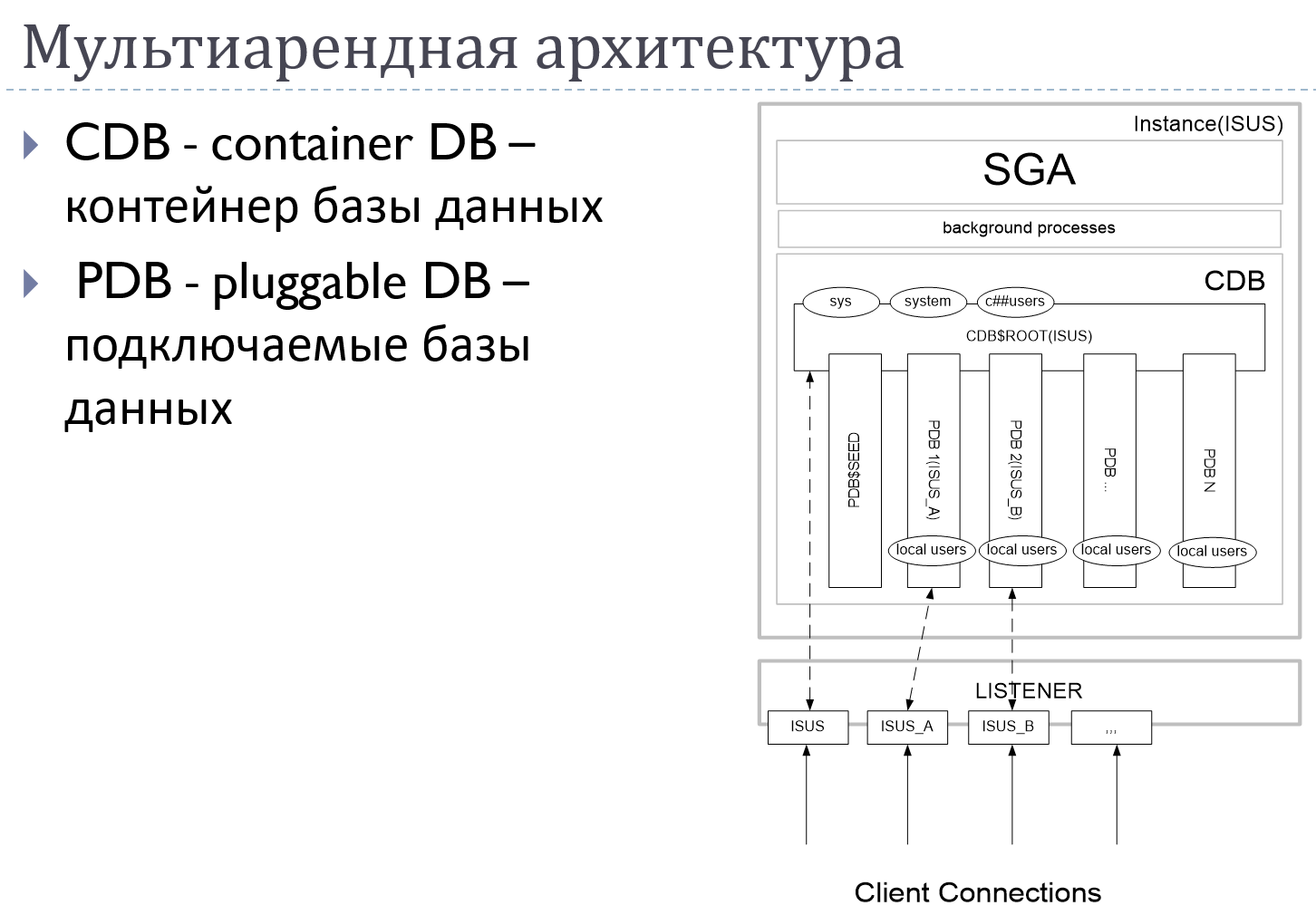
1. Мультиарендная архитектура Oracle Multitenant.

Oracle Multitenant - технология, позволяющая запустить несколько независимых баз данных в рамках одного экземпляра.

Каждая база данных имеет свой набор табличных пространств и набор схем, но при этом у них общая SGA и один набор серверных процессов.

Базы данных изолированы, друг о друге ничего не знают, не конфликтуют между собой.

Словарь разбивается на две части: общую часть и локальную.



Можно создавать несколько CDB – для разных версий программного обеспечения СУБД.

Одну и ту же PDB можно переносить между CDB.

В CDB создается главный контейнер Root. Root содержит метаданные CDB.

В одной CDB можно создать до 252 PDB.

Мультиарендная архитектура Oracle Multitenant была введена в Oracle Database 12c и представляет собой одну из самых значительных архитектурных изменений в истории Oracle. Она позволяет одной базе данных Oracle функционировать как контейнерная база данных (CDB), которая может содержать несколько подключаемых баз данных (PDB).

### Основные компоненты мультиарендной архитектуры

1. **Контейнерная база данных (CDB)**: Это основная база данных, которая содержит все необходимые компоненты для управления подключаемыми базами данных. CDB включает в себя корневой контейнер (CDB$ROOT), который содержит метаданные и общих пользователей.
2. **Подключаемая база данных (PDB)**: Это независимая база данных, которая может быть подключена к CDB. PDB содержит свои собственные схемы, объекты схем и несхемные объекты. PDB можно легко перемещать между различными CDB.
3. **Корневой контейнер (CDB$ROOT)**: Это контейнер, который содержит метаданные и общих пользователей, доступных для всех PDB в CDB.
4. **Системный контейнер**: Включает корневой контейнер и все PDB в CDB. Это логический контейнер для самой CDB.
5. **Приложенческий контейнер**: Состоит из одного корневого контейнера приложения и подключаемых баз данных приложения (PDB). Приложенческий контейнер позволяет управлять приложениями и их данными в рамках одной CDB.

### Преимущества мультиарендной архитектуры

* **Упрощенное управление**: Администраторы могут управлять несколькими PDB в рамках одной CDB, что упрощает администрирование и обновление.
* **Эффективное использование ресурсов**: Мультиарендная архитектура позволяет более эффективно использовать ресурсы сервера, так как несколько PDB могут совместно использовать ресурсы CDB.
* **Гибкость и масштабируемость**: PDB можно легко перемещать между различными CDB, что обеспечивает гибкость и масштабируемость.

### Применение мультиарендной архитектуры

Мультиарендная архитектура Oracle Multitenant используется для консолидации баз данных, упрощения управления и повышения эффективности использования ресурсов. Она особенно полезна для облачных сред и крупных корпоративных систем, где требуется управление множеством баз данных.

1. Файлы экземпляра Oracle. Файл параметров, управляющие файлы, файлы паролей, файлы трассировки.

**Файл параметров (Parameter File)**

Файл параметров предназначен для хранения параметров экземпляра базы данных Oracle. Существует два типа файлов параметров:

1. **PFILE (Parameter File)**: статичный, пользовательский текстовый файл, который редактируется стандартными текстовыми редакторами (например, "блокнот" или "vi"). Обычно этот файл находится на сервере, однако, для запуска Oracle с удаленной машины необходимо иметь локальную копию. АБД (Администраторы Баз Данных) обычно ссылаются на этот файл: INIT.ORA.
2. **SPFILE (Server Parameter File)**: с другой стороны, постоянно находящийся на сервере бинарный файл, который может быть изменен только с помощью команды "ALTER SYSTEM SET". Это значит, что больше нет необходимости хранить локальную копию файла параметров для запуска экземпляра с удаленной машины. Редактирование SPFILE-а вручную повредит его, после чего он станет неработоспособным.

**Управляющие файлы (Control Files)**

Control files – файлы, содержащие имена (местоположение) основных физических файлов базы данных и некоторых параметров. Используются для поиска других файлов операционной системы.

Местоположение управляющих файлов экземпляр получает из файла параметров. По умолчанию для надежности создается 2 управляющих файла. Можно создать больше. Обычно их размещают на разных дисковых носителях (для надежности).

**Файлы паролей (Password Files)**

Предназначен для аутентификации администраторов базы данных. (Обычно имеют расширение .ora. )

**Файлы трассировки (Trace Files)**

Файлы трассировки содержат диагностическую информацию и используются для отладки и анализа производительности базы данных. Они создаются Oracle автоматически при возникновении ошибок или по запросу администратора.

alert.log, trace\_file.trc

1. Файлы базы данных Oracle. Файлы данных, журналы, архивы.

**Файлы данных (Data Files)-?-не понимаю**

Файлы данных содержат фактические данные базы данных. Они хранятся в виде таблиц, индексов, кластеров и других объектов базы данных. Файлы данных могут быть организованы в табличные пространства (tablespaces), которые логически группируют данные.

users01.dbf, system01.dbf, example01.dbf

**Журналы (Redo Log Files)**

Журналы повторного выполнения - дисковые ресурсы, в которых фиксируются изменения вносимых пользователями в базу данных; журнал представляет собой файл операционной системы; как минимум должно быть два файла; журналы применяются при восстановлении базы данных.2

Журналы redo log содержат записи всех изменений, внесенных в базу данных. Они используются для восстановления базы данных в случае сбоя. Журналы redo log организованы в группы, и каждая группа содержит один или несколько файлов.

redo01.log, redo02.log, redo03.log

**Архивные журналы (Archived Redo Log Files)**

Если база данных работает в режиме ARCHIVELOG, перед перезаписью она сделает копию файлов оперативного журнала повтора. Затем эти архивированные файлы журнала повтора можно использовать для восстановления любой части базы данных в любой момент времени.

Архивные журналы создаются из redo log файлов после их заполнения. Они хранятся для обеспечения возможности восстановления базы данных до любого момента времени. Архивные журналы используются в стратегиях резервного копирования и восстановления.

arch\_001.log, arch\_002.log, arch\_003.log

1. Абстрактная модель Oracle. Логическая структура внешней памяти. !!!-без понятия

### Абстрактная модель Oracle

Абстрактная модель Oracle описывает, как данные организованы и управляются в базе данных Oracle. Она включает в себя несколько уровней абстракции:

1. **Физический уровень**: На этом уровне данные хранятся в виде файлов на диске. Физический уровень включает файлы данных, журналы redo log, архивные журналы и управляющие файлы.
2. **Логический уровень**: На этом уровне данные организованы в логические структуры, такие как табличные пространства, сегменты, экстенты и блоки данных.
3. **Схемный уровень**: На этом уровне данные организованы в схемы, которые включают таблицы, индексы, представления, процедуры и другие объекты базы данных.

### Логическая структура внешней памяти

Логическая структура внешней памяти в Oracle включает следующие компоненты:

1. **Табличные пространства (Tablespaces)**: Табличные пространства — это логические контейнеры для хранения данных. Каждое табличное пространство состоит из одного или нескольких файлов данных. Табличные пространства позволяют логически группировать данные и управлять их хранением.
2. **Сегменты (Segments)**: Сегменты — это логические структуры, которые хранят данные в табличных пространствах. Сегменты могут быть разных типов, таких как сегменты таблиц, сегменты индексов, сегменты кластеров и сегменты временных данных.
3. **Экстенты (Extents)**: Экстенты — это непрерывные области хранения данных внутри сегментов. Каждый сегмент состоит из одного или нескольких экстентов. Экстенты позволяют управлять выделением и использованием пространства для данных.
4. **Блоки данных (Data Blocks)**: Блоки данных — это наименьшие единицы хранения данных в Oracle. Каждый блок данных соответствует определенному количеству байтов на диске и содержит одну или несколько строк данных. Блоки данных являются основными единицами ввода-вывода в Oracle.
5. Абстрактная модель Oracle. Физическая структура внешней памяти. !!!-без понятия

**Абстрактная модель Oracle**

Абстрактная модель Oracle описывает, как данные организованы и управляются в базе данных Oracle. Она включает в себя несколько уровней абстракции:

1. **Физический уровень**: На этом уровне данные хранятся в виде файлов на диске. Физический уровень включает файлы данных, журналы redo log, архивные журналы и управляющие файлы.
2. **Логический уровень**: На этом уровне данные организованы в логические структуры, такие как табличные пространства, сегменты, экстенты и блоки данных.
3. **Схемный уровень**: На этом уровне данные организованы в схемы, которые включают таблицы, индексы, представления, процедуры и другие объекты базы данных.

**Физическая структура внешней памяти**

Физическая структура внешней памяти в Oracle включает следующие компоненты:

1. **Файлы данных (Data Files)**: Файлы данных содержат фактические данные базы данных. Они хранятся в виде таблиц, индексов, кластеров и других объектов базы данных. Файлы данных могут быть организованы в табличные пространства (tablespaces), которые логически группируют данные.

users01.dbf, system01.dbf, example01.dbf

1. **Журналы (Redo Log Files)**: Журналы redo log содержат записи всех изменений, внесенных в базу данных. Они используются для восстановления базы данных в случае сбоя. Журналы redo log организованы в группы, и каждая группа содержит один или несколько файлов.

redo01.log, redo02.log, redo03.log

1. **Архивные журналы (Archived Redo Log Files)**: Архивные журналы создаются из redo log файлов после их заполнения. Они хранятся для обеспечения возможности восстановления базы данных до любого момента времени. Архивные журналы используются в стратегиях резервного копирования и восстановления.

arch\_001.log, arch\_002.log, arch\_003.log

1. **Управляющие файлы (Control Files)**: Управляющие файлы содержат информацию о структуре базы данных, включая местоположение файлов данных, журнальных файлов и другую важную информацию. Управляющие файлы необходимы для запуска и работы базы данных.

control01.ctl, control02.ctl, control03.ctl

1. Абстрактная модель Oracle. Структура SGA.

SGA – System Global Area – часть выделенной для Oracle ОЗУ, которую разделяют между собой все серверные процессы (в том числе и фоновые). Необходима для ускорения производительности запросов и обеспечения большого объема параллельной активности.

Значительно важнее, чем PGA, особенно в крупных OLTP-базах; Крайне важно при администрировании БД верно указать размер SGA, ведь в ней будет храниться много данных, т.к. хранить данные в ОЗУ (в SGA) гораздо выгоднее и быстрее, чем сразу отправлять их на диск и читать их с диска.

При запуске инстанса в ОЗУ выделяется некоторый объем памяти для Oracle, определенный размером SGA (его размер прописан в конфигурационном файле). При остановке инстанса выделенная SGA память возвращается системе.

### Структура SGA (System Global Area)

SGA — это область памяти, выделенная для экземпляра базы данных Oracle. Она содержит данные и управляющую информацию, необходимые для работы базы данных. Память в SGA разделяемая, т.е. делится между основными пулами, перечисленными ниже.

Основные пулы памяти SGA:

* Java pool – куча для создания объектов Java. Нужен для работы Java-машины.
* Large pool – большой пул – крупные выделения памяти. Необходим только при параллельных запросах или при архитектуре разделяемого сервера. Не поддерживает механизм LRU. Память освобождается сразу после использования (как в recycle-пуле).
* Shared pool – разделяемый пул – кэш для общего для всех юзеров кода. Хранит исполняемый код PL/SQL и операторы SQL, а также информацию, относящуюся к словарю базы данных (библиотечный кэш, кэш словаря, разделяемая область SQL).
* Log buffer – буфер журналов повтора – для восстановления изменений после DML-операторов. Не превышает пары мегабайт. Содержимое буфера пишется на диск: 1) через 3 секунды; 2) при коммите транзакции; 3) при заполнении буфера на одну треть; 4) если в буфере больше 1 МБ данных.
* Buffer cache – буферный пул (кэш) – буферы памяти, использующиеся вместо чтения инфы с диска. Подробнее описано ниже.
* Streams pool – пул потоков – содержит средство Oracle Streams для репликации данных.
* Также в SGA есть фиксированная область (по сути метаданные), библиотечный кэш и кэш результатов.

PGA – Program Global Area – неразделяемая память, для каждого серверного и фонового процесса будет выделяться своя личная область PGA. Содержат управляющую конфигурационную информацию для процессов. Доступ к PGA имеет только соответствующий процесс. Записывать сюда может только код Oracle.

1. Абстрактная модель Oracle. Серверные процессы Oracle.!!!-не уверен

Серверные процессы Oracle

Серверный процесс в Oracle представляет собой фоновый процесс, который управляет подключениями клиентов к базе данных и обрабатывает запросы, поступающие от клиентских приложений. Каждое активное подключение к базе данных представлено отдельным серверным процессом, который обеспечивает выполнение операций, отправку результатов запросов обратно клиенту и обработку других задач, связанных с поддержанием соединения с базой данных.

SELECT SID, SERIAL#, USERNAME, PROGRAM, STATUS

FROM V$SESSION;

SHOW PROCESSLIST

1. Абстрактная модель Oracle. Фоновые процессы Oracle.

**Ф**оновые (background) процессы – специальная группа процессов для обеспечения производительности и поддержки работы большого числа пользователей

* **DBWn (Database Writer Process):**Фоновый процесс записывающий по LRU измененные блоки (грязные блоки) в файлы базы данных.
  + В большинстве случаев достаточно одного процесса
  + Если процессор 1, то в более чем в 1 DBW не смысла
  + Проверяет с периодичностью не менее 3 сек. наличие измененных блоков и инициирует checkpoint и по команде фонового процесса **CKPT** сбрасывает некоторую их часть на диск
  + Использует асинхронный ввод-вывод для записи блоков на диск
* **LGWR (Log Writer Process):**
  + Только один
  + Управляет буфером журналов повтора
  + Записывает блоки буфера журналов повтора в группы журналов
  + Записывает изменения базы данных до их фиксации DBWR в базе данных
  + Переключает текущую группу
    - Раз в 3 секунды
    - При фиксации транзакции
    - При заполнении буфера на 1/3 или записи в него 1Мб
* **SMON (System Monitor Process):**
  + Восстановление экземпляра для узла
  + Восстановление незавершенных транзакций
  + Очистка временных сегментов данных
  + Очистка временных табличных пространств
  + Объединение свободного пространства
  + Очистка таблицы OBJ
  + Сжатие сегментов отката
* **PMON (Process Monitor Process):**
  + Отвечает за очистку после ненормального закрытия подключений
  + Инициирует откат незафиксированных транзакций, снятие блокировок, и освобождение ресурсов SGA
  + Следит за работой других фоновых процессов, отвечает за их перезапуск
  + Восстанавливает работу dispatcher или shadow процессов при системном сбое
* **ARCn (Archiver Process):**  
  Архивирует файлы журнальных изменений для обеспечения возможности восстановления данных (является необязательным).
* **RECO (Recovery Process):**  
  Разрешение проблем связанных с распределенными транзакциями
  + Одна из баз данных, к которой первоначально подключился клиент, становится координатором
  + Сервер опрашивает остальные N -1 серверов, готовы ли они фиксировать транзакцию
  + Каждый из N -1 серверов сообщает о своем состоянии готовности как да (YES) или нет (NO)
  + Если любой из серверов вернул NO, вся транзакция откатывается
  + Если все серверы вернули YES, координатор рассылает всем N - 1 серверам сообщение о постоянной фиксации
* **LREG (Listener Registration Process):**

Периодическая регистрация сервисов в процессе Listener

* **CKPT (checkpoint):**
  + Выполняет процесс checkpoint
  + Выполняется при shutdown, alter system checkpoint, переключении REDO – журнала, периодическом сообщение от DWR, backup
  + Записывает информацию о контрольной точке в управляющие файлы
  + Дает команду DBW на сброс буферов
  + Дает команду LGWR на сброс буферов
* FBDA (Flashback Data Archiever):

Архивирование ретроспективных данных

(лекция 6)

1. Процесс-слушатель Oracle и его основные параметры.

Oracle Net Listener – процесс на стороне сервера, прослушивающий входящие запросы клиента на соединение с экземпляром.

Listener – это программа-сервер, прослушивающая TCP-порт, принимающая запросы на соединение с Oracle экземпляром от программ-клиентов.

В результате успешной работы Listener устанавливается соединение между программой-клиентом и обработчиком запросов экземпляра.

По умолчанию TCP-порт 1521

Экземпляр может иметь несколько точек подключения

Точки подключения называются сервисами и имеют символические имена

При инсталляции автоматически создается два сервиса:

SYS$USERS (по умолчанию, указывается SID в параметрах соединения),

сервис с именем инстанса (указывается сервис)

При создании PDB для нее автоматически добавляется сервис с именем, совпадающим с PDB-именем

**Старт экземпляра**

фоновый процесс LREG считывает имена и параметры зарегистрированных сервисов экземпляра

регистрирует их в Listener

если Listener не обнаружен, то попытки регистрации периодично повторяются

Listener прослушивает запросы к сервисам

**Запрос на соеденение**

1)клиент выполняет запрос к Listener на соединение с сервисом экземпляра

2) Listener запрашивает соединение с сервером

3)сервер возвращает параметры соединения с обработчиком сервиса

4) Listener сообщает параметры соединения клиенту

5) клиент соединяется с обработчиком запросов сервиса для дальнейшей работы с сервисом в рамках соединения

lsnrctl – утилита управления процессом Listener

(картинки можно посмотреть в лекции 6)

1. Сетевые настройки Oracle. Установление соединения по сети.

Oracle использует сетевой компонент Oracle Net для управления сетевыми соединениями между клиентами и базой данных. Oracle Net предоставляет механизм для передачи данных через различные сетевые протоколы.

**Основные компоненты**

1. Listener:

- Listener (Слушатель) является серверным процессом, который прослушивает запросы на соединение с базой данных.

- Конфигурационный файл слушателя называется “listener.ora”.

2. TNSNAMES.ORA:

- Клиентский файл конфигурации, который содержит сведения о службах базы данных, таких как имя хоста, порт и имя службы.

- Используется для упрощения процесса подключения клиентов к базе данных.

3. SQL Net:

- Протокол, используемый для обмена данными между клиентом и сервером базы данных.

**Установление соединения по сети**

1. Настройка слушателя:

- Отредактируйте файл “listener.ora” для конфигурации слушателя, указав порт и имя службы.

2. Настройка TNSNAMES.ORA:

- Отредактируйте файл “tnsnames.ora”, чтобы указать параметры соединения для клиентов.

3. Запуск слушателя:

- Используйте команду “lsnrctl start” для запуска процесса слушателя.

4. Подключение клиента:

- Клиент использует утилиту, такую как SQL Plus, для подключения к базе данных, используя параметры, указанные в файле “tnsnames.ora”.

- Пример команды подключения:

sqlplus username/password@YOUR\_DB

Эти шаги помогут вам настроить сетевые параметры и установить соединение с базой данных Oracle по сети.

(лекция 7)

1. Табличные пространства СУБД Oracle и их основные параметры.

Табличное пространство – логическая структура хранения данных, контейнер сегментов

С одним табличным пространством связаны один или несколько файлов операционной системы, с каждым файлом связано только одно табличное пространство

* PERMANENT - хранение постоянных объектов БД
  + Пространств может быть несколько
  + Можно создавать свои объекты в любом из PERMANENT пространств, если это не запрещено
  + Одно может приписано в качестве пространства по умолчанию.
* TEMPORARY - хранение временных данных
  + Пространств может быть несколько
  + Пространство приписывается одному или нескольким пользователям, которые могут там размещать свои временные данные
* UNDO - хранение сегментов отката
  + Пространств может быть несколько, но активным является всегда только одно (указывается в конфигурационном файле SPIFLE.ORA)

(лекция 2)

1. Роли и привилегии СУБД Oracle и их основные параметры.

Роль – это именованный набор привилегий, который может быть назначен пользователю или другой роли.

1. Создание ролей:

- Роли могут быть созданы с помощью команды “CREATE ROLE”.

2. Назначение привилегий роли:

- Привилегии могут быть назначены роли с помощью команды “GRANT”.

3. Назначение ролей пользователям:

- Роли могут быть назначены пользователям также с помощью команды “GRANT”.

Привилегия - это право выполнения определенных действий в базе данных. Привилегии делятся на две основные категории: системные и объектные.

1. Системные привилегии:

- Это права на выполнение административных задач или системных операций.

- Примеры: “CREATE TABLE”, “CREATE USER”, “DROP DATABASE”.

2. Объектные привилегии:

- Это права на выполнение операций над конкретными объектами базы данных, такими как таблицы, представления и последовательности.

- Примеры: “SELECT”, “INSERT”, “UPDATE”, “DELETE” на таблицу или представление.

Основные параметры привилегий

1. Назначение привилегий:

- Привилегии могут быть назначены пользователям или ролям с помощью команды “GRANT”.

2. Отзыв привилегий:

- Привилегии могут быть отозваны с помощью команды “REVOKE”.

3. Передача привилегий:

- Некоторые привилегии могут быть переданы другим пользователям с использованием опции “WITH GRANT OPTION”.

1. Пользователь СУБД Oracle и его основные параметры.

Пользователь в СУБД Oracle представляет собой учетную запись, которая позволяет человеку или приложению взаимодействовать с базой данных

Основные параметры пользователя

1. Имя пользователя (Username):

- Уникальный идентификатор пользователя в базе данных.

2. Пароль (Password):

- Секретная фраза, которая используется для аутентификации пользователя при подключении к базе данных.

- Пароль можно задать при создании пользователя или изменить позже.

3. Дефолтное табличное пространство (Default Tablespace):

- Табличное пространство, в которое по умолчанию будут создаваться объекты пользователя.

4. Временное табличное пространство (Temporary Tablespace):

- Табличное пространство, используемое для временных объектов, таких как временные таблицы и индексы.

5. Профиль (Profile):

- Набор параметров, который позволяет управлять ресурсами и политикой безопасности для пользователей.

6. Квоты (Quotas):

- Ограничения на объем пространства, который пользователь может использовать в конкретном табличном пространстве.

Пример создания пользователя

Полный пример создания пользователя и назначения ему необходимых параметров и привилегий:

--sql

CREATE USER user\_name IDENTIFIED BY password

DEFAULT TABLESPACE tablespace

TEMPORARY TABLESPACE temp

PROFILE default;

ALTER USER user\_name QUOTA 100M ON tablespace;

GRANT CREATE SESSION TO user\_name;

GRANT SELECT, INSERT ON employees TO user\_name;

“““

1. Профиль безопасности СУБД Oracle и его основные параметры.

Профиль безопасности в СУБД Oracle представляет собой набор параметров, которые управляют ресурсами и обеспечивают политику безопасности для пользователей базы данных. Он позволяет ограничить использование системных ресурсов и применить определенные правила безопасности для различных категорий пользователей.

**Основные параметры профиля безопасности**

1. LIMIT: Определяет ограничения на использование ресурсов для пользователей, которым назначен профиль.

- Примеры:

- “SESSIONS\_PER\_USER”: Максимальное количество одновременных сеансов для одного пользователя.

- “CPU\_PER\_SESSION”: Максимальное количество времени процессора, которое может быть использовано в одном сеансе.

2. PASSWORD MANAGEMENT: Управляет правилами безопасности паролей.

- Примеры:

- “PASSWORD\_LIFE\_TIME”: Максимальное время действия пароля, после чего требуется его смена.

- “PASSWORD\_REUSE\_TIME”: Минимальное время, которое должно пройти до повторного использования старого пароля.

- “PASSWORD\_VERIFY\_FUNCTION”: Функция, используемая для проверки сложности пароля.

3. IDLE TIME: Управляет временем неактивности.

- Пример:

- “IDLE\_TIME”: Максимальное время неактивности (в минутах) перед автоматическим завершением сеанса.

**Примеры создания и настройки профиля**

1. Создание профиля с ограничениями на использование ресурсов и управлением паролями:

“““sql

CREATE PROFILE secure\_profile LIMIT

SESSIONS\_PER\_USER 3

CPU\_PER\_SESSION 10000

PASSWORD\_LIFE\_TIME 60

PASSWORD\_REUSE\_TIME 180

PASSWORD\_VERIFY\_FUNCTION verify\_function

IDLE\_TIME 30;

“““

2. Назначение профиля пользователю:

“““sql

ALTER USER user\_name PROFILE secure\_profile;

“““

1. Таблица в СУБД Oracle и ее основные параметры. Типы данных базы данных. Ограничения целостности в таблицах.

Таблицы в СУБД Oracle являются основными объектами для хранения данных. Они состоят из строк и столбцов, где каждый столбец имеет определенный тип данных. Давайте рассмотрим основные параметры таблиц, типы данных и ограничения целостности в таблицах.

**Основные параметры таблиц**

1. Имя таблицы:

- Уникальное имя, которое идентифицирует таблицу в базе данных.

2. Столбцы:

- Определяют структуру таблицы, каждый столбец имеет имя и тип данных.

3. Типы данных:

- Определяют тип значений, которые могут быть сохранены в столбце.

**Типы данных базы данных**

1. Числовые данные:

- NUMBER(n, s)

* Числовой тип переменной длины
* Точность n <= 38, общее количество цифр
* Масштаб s = [-84,127], количество цифр после запятой

2. Даты и время:

|  |  |
| --- | --- |
| DATE | 7 байтовое поле фиксированной длины, используемое для хранения даты и времени |
| INTERVAL  DAY TO SECOND | 11 байтовое поле фиксированной длины для интервала времени: Дни, часы, минуты, секунды |
| INTERVAL  YEAR TO MONTH TIMESTAMP | 5 байтовое поле фиксированной длины для интервала времени: Годы и месяцы |
| TIMESTAMP  WITH TIME ZONE | 13 байтовое поле фиксированной длины: Дата, время и настройки, связанные с часовым поясом. |
| TIMESTAMP  WITH LOCAL TIME | 7-11 байтовое поле переменной длины: Дата и время, приведенные к часовому поясу базы данных |

3. Символьные:

|  |  |
| --- | --- |
| LONG | Символьный, переменной длины, до 2GB, оставлен для совместимости |
| RAW(n) | Переменной длины, для бинарных данных  n <= 2000 byte  оставлен для совместимости |
| LONG RAW | Бинарные данные до 2GB |
| CLOB | Символьный тип большой объект до 4GB |
| NLOB | CLOB для многобайтных символов |
| BLOB | Большой двоичный объект до 4GB |
| BFILE | Указатель на двоичный файл операционной системы |

**Ограничения целостности в таблицах**

Ограничения целостности в таблицах в Oracle используются для обеспечения корректности и согласованности данных. Они позволяют задать правила для значений в столбцах таблиц и связей между таблицами. Основные типы ограничений целостности включают:

1. PRIMARY KEY (Первичный ключ):

- Гарантирует уникальность значений в столбце или наборе столбцов.

- В таблице может быть только один первичный ключ.

2. FOREIGN KEY (Внешний ключ):

- Обеспечивает ссылочную целостность, указывая, что значения в одном столбце или наборе столбцов должны соответствовать значениям в другой таблице.

3. UNIQUE (Уникальное ограничение):

- Обеспечивает уникальность значений в одном или нескольких столбцах.

- В отличие от первичного ключа, может быть несколько уникальных ограничений в одной таблице.

4. CHECK (Проверка):

- Определяет условие, которому должны соответствовать значения в столбце.

5. NOT NULL (Не NULL):

- Обеспечивает, что столбец не может содержать значения NULL.

6. DEFAULT (Значение по умолчанию):

- Устанавливает значение по умолчанию для столбца, если значение не указано при вставке данных.

(лекция 8)

1. Временные таблицы СУБД Oracle.

Временные таблицы – специальные типы таблиц, которые используются для хранения данных, временно создаваемых в ходе выполнения транзакций.

Состоит из столбцов и строк, как и обычная таблица

* Временные таблицы – глобальны
* Привилегии для создания временной таблицы CREATE TABLE
* Можно разместить временную таблицу в заданном табличном пространстве.
* Временные таблицы – это шаблон, хранящийся в словаре базы данных, для нее выделяется временный сегмент в (по умолчанию) TEMPORARY-табличном пространстве и для каждого пользователя свой.
* Каждый пользователь видит только свои данные (свой сегмент данных).
* Статичны: временные таблицы создаются (CREATE) один раз и существуют, пока их не удалят (DROP).
* DROP не получится, если таблица в этот момент используется другим пользователем.
* Временные таблицы бывают:
  + ON COMMIT PRESERVE ROWS – на время сеанса, данные существуют только на время сеанса, возможны все DML-операторы, TCL-операторы
  + ON COMMIT DELETE ROWS – на время транзакции, данные существуют только на время транзакции, возможны все DML-операторы, после выполнения COMMIT или ROLLBACK таблица становится пустой
* В начале сеанса временная таблица всегда пуста
* Для временных таблиц можно создавать триггеры
* Для временных таблиц можно указать констрейны (ограничения)
* Для временных таблиц можно создавать индексы
* Не могут быть индексно-организованными, нельзя секционировать, размещать в кластере.
* Данные повторного выполнения генерируются, но их количество пренебрежительно мало.

(лекция 8)

1. Индексы базы данных СУБД Oracle. Виды и особенности применения индексов.

Индекс – структура базы данных, используемая сервером для быстрого поиска строки в таблице

* Типы индексов:
  + Табличный (B\*Tree) индекс
    - Табличный индекс (B\*Tree) структурирован в виде сбалансированного дерева
    - Листовой блок содержит индексированные значения столбца и соответствующий ему идентификатор строки (RowId)
    - Предназначен для индексирования уникальных столбцов или столбцов с высокой селективностью
  + Битовый индекс
    - Битовый индекс создает битовые карты для каждого возможного значения столбца, где каждому биту соответствует строка, а значение бита 1 (0) означает, что соответствующая строка содержит (не содержит) индексируемое значение
    - Предназначен для индексирования столбцов с низкой селективностью
    - Не подходит для таблиц с частым обновлением
    - Хорошо подходят для хранилищ данных
  + Функциональный индекс
    - Функциональный индекс – предварительно вычисляют значения функции по заданному столбцу и сохраняют результат в индексе
    - LOWER(NAME) / UPPER (NAME)

(лекция 8)

1. Последовательность СУБД Oracle и ее параметры.

Последовательность (Sequence) в СУБД Oracle — это объект базы данных, используемый для генерации уникальных числовых значений. Последовательности часто используются для создания значений первичных ключей. Давайте рассмотрим основные параметры последовательностей и их использование.

Основные параметры последовательностей

1. START WITH:

- Определяет начальное значение последовательности.

2. INCREMENT BY:

- Определяет шаг, на который увеличивается значение последовательности.

3. MAXVALUE:

- Определяет максимальное значение последовательности.

4. MINVALUE:

- Определяет минимальное значение последовательности.

5. CYCLE / NOCYCLE:

- Определяет, должна ли последовательность начинать с начального значения после достижения максимального значения (CYCLE) или нет (NOCYCLE).

6. CACHE / NOCACHE:

- Определяет количество предварительно сгенерированных значений, которые сохраняются в памяти для быстрого доступа (CACHE) или отключает кэширование (NOCACHE).

7. ORDER / NOORDER:

- Определяет, должны ли значения последовательности генерироваться в порядке (ORDER) или можно генерировать значения в любом порядке (NOORDER).

**Пример**

CREATE SEQUENCE seq\_example

START WITH 1

INCREMENT BY 1

MAXVALUE 1000

CYCLE

CACHE 20;

2. Получение следующего значения последовательности:

- Используйте псевдоколонку “NEXTVAL” для получения следующего значения последовательности.

- Пример:

SELECT seq\_example.NEXTVAL FROM dual;

3. Получение текущего значения последовательности:

- Используйте псевдоколонку “CURRVAL” для получения текущего значения последовательности.

- Пример:

SELECT seq\_example.CURRVAL FROM dual;

1. Кластер и его параметры.

Кластер – объект БД, который хранит значения общих столбцов нескольких таблиц

Основные параметры кластера

1. Имя кластера:

- Уникальное имя, которое идентифицирует кластер в базе данных.

2. Колонка ключа кластера:

- Столбец или несколько столбцов, которые используются для группировки строк.

3. SIZE:

- Определяет размер блока кластера (количество байт, выделенное под каждый кластерный ключ).

Преимущества использования кластеров

- Уменьшение операций ввода-вывода: Снижение числа операций ввода-вывода при доступе к связанным данным.

- Улучшение производительности: Повышение производительности запросов, часто использующих соединения (JOIN).

- Эффективное использование пространства: Более компактное хранение связанных данных.

**Пример**

CREATE CLUSTER emp\_сluster (department\_id NUMBER) SIZE 512;

(лекция 8)

1. Представление в СУБД Oracle и его параметры.

Представление (View) в СУБД Oracle представляет собой виртуальную таблицу, созданную на основе результатов запроса. Представления используются для упрощения доступа к данным, предоставления уровня абстракции и обеспечения безопасности.

Основные параметры представлений

1. Имя представления:

- Уникальное имя, которое идентифицирует представление в базе данных.

2. Запрос для создания представления:

- Представление создается на основе SQL-запроса, который определяет, какие данные будут доступны через представление.

3. READ ONLY:

- Опция, которая делает представление доступным только для чтения, предотвращая любые изменения данных через это представление.

4. FORCE / NOFORCE:

- Опция FORCE позволяет создать представление, даже если базовые таблицы не существуют на момент создания представления.

Преимущества использования представлений

- Упрощение доступа к данным: Представления позволяют пользователям работать с данными, не зная подробностей структуры базовых таблиц.

- Повышение безопасности: Представления могут ограничивать доступ к определенным данным, предоставляя только необходимые столбцы или строки.

- Абстракция и независимость данных: Представления обеспечивают уровень абстракции, который позволяет изменять базовые таблицы без изменения запросов пользователей.

Представления являются мощным инструментом в СУБД Oracle для упрощения управления данными и обеспечения безопасности.

-- Создание представления emp\_view на основе запроса, только для чтения

CREATE FORCE VIEW emp\_view AS

SELECT employee\_id, first\_name, last\_name

FROM table

WHERE department\_id = 10

WITH READ ONLY;

1. Материализованное представление и его параметры.

Материализованное представление (Materialized View) в СУБД Oracle — это объект базы данных, который сохраняет результаты запроса в виде физической копии данных. Материализованное представление используется для повышения производительности и быстрого доступа к данным, особенно в случае сложных запросов и больших объемов данных. В отличие от обычного представления (View), материализованное представление хранит данные, а не только запрос.

Основные параметры материализованного представления

1. SELECT:

- Запрос, определяющий данные, которые будут сохранены в материализованном представлении.

2. BUILD IMMEDIATE / BUILD DEFERRED:

- BUILD IMMEDIATE: Материализованное представление создается и заполняется данными сразу после выполнения команды CREATE.

- BUILD DEFERRED: Материализованное представление создается, но данные заполняются позже, при первой синхронизации.

3. REFRESH:

- Определяет, как и когда данные в материализованном представлении будут обновляться.

- REFRESH ON COMMIT: Обновление происходит при фиксации транзакции.

- REFRESH ON DEMAND: Обновление происходит по запросу.

- REFRESH FAST / COMPLETE: Определяет, будет ли обновление инкрементальным (FAST) или полным (COMPLETE).

4. START WITH / NEXT:

- Параметры, определяющие расписание обновления.

- START WITH: Дата и время первого обновления.

- NEXT: Интервал обновлений.

Примеры создания и использования материализованных представлений

1. Создание материализованного представления с немедленной сборкой данных:

CREATE MATERIALIZED VIEW mv\_example

BUILD IMMEDIATE

AS SELECT employee\_id, first\_name, last\_name FROM employees;

Преимущества материализованных представлений

- Улучшение производительности: Быстрый доступ к данным благодаря сохранению результатов запроса.

- Снижение нагрузки на базу данных: Уменьшение числа сложных запросов к базовым таблицам.

- Автоматическое обновление: Автоматизация процесса обновления данных с использованием различных режимов обновления.

1. Частные и публичные синонимы СУБД Oracle.

Синоним – способ обращаться к объекту базы данных без указания обязательной полной идентификации объекта (хост – экземпляр – владелец – объект).

* Частный синоним принадлежит пользователю, который его создал.
* Публичный синоним используется совместно всеми пользователями базы данных.

Для создания синонима нужна привилегия CREATE (PUBLIC) SYNONYM

Синоним может указывать на:

* Таблицы,
* Процедуры,
* Функции,
* Последовательности,
* Представления
* Пакеты
* Объекты в локальной или удаленной базе данных

(лекция 8)

1. Структура программы языка PL/SQL. Анонимные и именованные блоки.

Procedural Language

**Запись в виде:**

Заголовок

Is

Раздел объявления

Begin

Исполняемый раздел

Exception

Раздел исключений

End;

**Анонимные блоки:**

* Не имеет секции заголовка
* Используется как скрипт для выполнения PL/SQL выражений
* Не может быть вызван из другого блока
* Начинается с DECLARE или BEGIN
* Варианты использования:
  + Триггер на стороне клиента (Oracle Development Tools)
  + Триггер базы данных (содержит АБ)
  + SQL-скрипт (описание процедур, функций и execute)
  + Откомпилированная программа (блок в execute команде, выполняющейся на сервере)

**Именованные блоки:**

Именованные блоки в PL/SQL в Oracle — это программные блоки, которым присваивается имя, что позволяет вызывать их в других частях программы. К именованным блокам относятся процедуры, функции, пакеты и триггеры

(лекция 10)

1. Типы данных, основные операции, константы языка PL/SQL. Псевдостолбцы.

**Типы данных в PL/SQL**

PL/SQL поддерживает множество типов данных, включая:

1. Символьные данные:

- “CHAR(size)”: Фиксированная длина символьных данных.

- “VARCHAR2(size)”: Переменная длина символьных данных.

- “CLOB”: Большой объект символьных данных.

2. Числовые данные:

- “NUMBER(p, s)”: Число с точностью “p” и шкалой “s”.

- “BINARY\_INTEGER”: Целое число, использующее двоичную арифметику.

- “PLS\_INTEGER”: Оптимизированный для производительности тип целых чисел.

3. Даты и время:

- “DATE”: Хранение даты и времени.

- “TIMESTAMP”: Хранение даты и времени с большей точностью.

4. Двоичные данные:

- “BLOB”: Большой двоичный объект.

- “RAW”: Двоичные данные переменной длины.

5. Булевы данные:

- “BOOLEAN”: Истина или ложь (TRUE, FALSE, NULL).

**Основные операции в PL/SQL**

PL/SQL поддерживает стандартные арифметические, логические и сравнения операции:

1. Арифметические операции:

- Сложение: “+”

- Вычитание: “-”

- Умножение: “\*”

- Деление: “/”

- Остаток от деления: “MOD”

2. Логические операции:

- И: “AND”

- ИЛИ: “OR”

- НЕ: “NOT”

3. Операции сравнения:

- Равно: “=“

- Не равно: “!=“ или “<>“

- Меньше: “<“

- Больше: “>“

- Меньше или равно: “<=“

- Больше или равно: “>=“

**Константы в PL/SQL**

Константы в PL/SQL объявляются с помощью ключевого слова “CONSTANT”. Значение константы назначается один раз и не может быть изменено.

**Псевдостолбцы**

Псевдостолбцы в Oracle представляют собой специальные значения, которые не хранятся в таблицах, но могут использоваться в SQL-запросах и операторах PL/SQL. Некоторые из наиболее часто используемых псевдостолбцов:

1. CURRVAL и NEXTVAL:

- Используются для работы с последовательностями.

- “CURRVAL”: Текущее значение последовательности.

- “NEXTVAL”: Следующее значение последовательности.

2. ROWNUM:

- Номер строки в результатах запроса.

- Используется для ограничения числа возвращаемых строк.

3. ROWID:

- Уникальный идентификатор строки в таблице.

- Используется для быстрого доступа к строкам.

4. LEVEL:

- Уровень строки в иерархическом запросе.

5. SYSDATE:

- Текущая дата и время сервера базы данных.

(лекция 10)

1. Поддержка национальных языков в СУБД Oracle. Наборы символов. Байтовая и символьная семантика символов.

Поддержка национальных языков в СУБД Oracle позволяет разработчикам и администраторам баз данных эффективно работать с текстами на различных языках и использовать разнообразные наборы символов для глобальной доступности базы данных.

**Поддержка национальных языков**

Oracle предоставляет множество инструментов и функций для поддержки национальных языков, обеспечивая возможность работы с многоязычными данными. Это достигается за счет использования национальных наборов символов и функций для обработки текстов на разных языках.

Наборы символов

Наборы символов в Oracle определяют, какие символы могут быть использованы в базе данных и как они кодируются. Oracle поддерживает множество наборов символов, включая:

1. ASCII: Стандартный набор символов, включающий английские буквы, цифры и некоторые специальные символы.

2. ISO 8859-1 (Latin-1): Набор символов для западноевропейских языков.

3. UTF-8: Универсальный набор символов для кодирования символов Unicode, поддерживающий множество языков и символов.

4. AL32UTF8: Расширенная версия UTF-8, используемая в Oracle для полной поддержки Unicode.

Байтовая и символьная семантика символов

Oracle позволяет использовать как байтовую, так и символьную семантику для обработки данных. Эти две семантики определяют, как интерпретируются и обрабатываются символы в строках:

1. Байтовая семантика (BYTE semantics):

- В байтовой семантике длина строки измеряется в байтах.

- Это полезно при работе с наборами символов фиксированной длины, таких как ASCII.

2. Символьная семантика (CHAR semantics):

- В символьной семантике длина строки измеряется в символах.

- Это полезно при работе с наборами символов переменной длины, такими как Unicode.

1. Связанные объявления переменных: инструкция %TYPE, инструкция %ROWTYPE.

В PL/SQL в Oracle связанное объявление переменных позволяет определить тип данных переменной или запись на основе существующих структур данных, таких как столбцы таблиц или строки. Для этого используются инструкции “%TYPE” и “%ROWTYPE”.

Инструкция %TYPE

Инструкция “%TYPE” используется для объявления переменной, тип которой соответствует типу данных столбца таблицы или атрибуту другого объекта. Это позволяет обеспечить соответствие типов данных и избежать ошибок при изменении типов данных в таблицах.

Инструкция %ROWTYPE

Инструкция “%ROWTYPE” используется для объявления записи (record), тип которой соответствует строке из таблицы или курсора. Это позволяет работать с полным набором данных строки, не определяя каждый столбец отдельно.

Преимущества использования %TYPE и %ROWTYPE

1. Соответствие типов данных: Обеспечивает автоматическое соответствие типов данных переменных и таблиц, что уменьшает вероятность ошибок.

2. Упрощенное сопровождение кода: Изменения в структурах таблиц автоматически отражаются в коде PL/SQL, что упрощает его сопровождение и обновление.

3. Читабельность кода: Улучшает читабельность кода, так как типы данных определяются явно и согласованно.

1. Локальные процедуры и функции языка PL/SQL.

Процедуры и функции в СУБД Oracle являются основными строительными блоками языка PL/SQL, позволяющими организовать и структурировать код, повторно использовать логику и инкапсулировать операции. Оба этих объекта представляют собой именованные программные блоки, но у них есть важные различия в том, как они используются и что они делают.

Локальная процедура — это процедура, которая определена внутри другого блока PL/SQL. Она может быть вызвана только изнутри этого блока.

Локальная функция — это функция, которая определена внутри другого блока PL/SQL. Она может быть вызвана только изнутри этого блока и может возвращать значение.

Преимущества локальных процедур и функций

1. Организация кода: Локальные процедуры и функции помогают организовать и структурировать код, делая его более логичным и легко читаемым.

2. Повторное использование кода: Локальные процедуры и функции позволяют повторно использовать одну и ту же логику внутри блока, избегая дублирования кода.

3. Инкапсуляция логики: Локальные процедуры и функции инкапсулируют логику, которая используется только внутри определенного блока, что делает код более модульным и независимым.

(лекции 11, 12)

1. Использование записей в PL/SQL. Вложенные записи.

Записи (record) в PL/SQL представляют собой составные типы данных, которые позволяют группировать связанные значения различных типов данных в одну структуру. Использование записей помогает организовать и управлять данными в более структурированной и удобной форме. Вложенные записи — это записи, которые содержат другие записи в качестве своих полей.

Записи в PL/SQL могут быть объявлены на основе существующей таблицы с использованием инструкции “%ROWTYPE” или как пользовательские типы данных.

Преимущества использования записей и вложенных записей

1. Организация данных: Записи позволяют группировать связанные данные в одну структуру, что упрощает управление и обработку данных.

2. Модульность и повторное использование: Записи и вложенные записи могут быть объявлены как пользовательские типы данных и использоваться в различных частях программы.

3. Улучшение читаемости кода: Использование записей делает код более понятным и структурированным.

1. Операторы управления, операторы цикла языка PL/SQL.

Операторы управления и операторы цикла в языке PL/SQL используются для управления потоком выполнения программы и выполнения повторяющихся операций. Давайте рассмотрим основные операторы управления и циклов в PL/SQL.

Операторы управления

1. IF-THEN-ELSE:

- Оператор “IF-THEN-ELSE” позволяет выполнять условные проверки и управлять выполнением кода в зависимости от результата этих проверок.

2. CASE:

- Оператор “CASE” позволяет выполнять множественные условные проверки и управлять выполнением кода на основе различных условий.

Операторы цикла

1. LOOP:

- Простой цикл, который выполняет блок операторов до тех пор, пока явно не выйдет из цикла с помощью оператора “EXIT”.

2. WHILE-LOOP:

- Цикл “WHILE-LOOP” выполняет блок операторов до тех пор, пока условие истинно.

3. FOR-LOOP:

- Цикл “FOR-LOOP” выполняет блок операторов для определенного диапазона значений.

1. Курсоры. Виды курсоров. Схемы обработки курсора. Атрибуты курсора. Курсоры с параметрами. Динамические курсоры.

Курсоры в PL/SQL используются для обработки множества строк, полученных в результате выполнения SQL-запроса. Они позволяют выполнять операции с каждым результатом запроса поочередно. Давайте рассмотрим различные аспекты работы с курсорами в Oracle.

**Виды курсоров**

1. Явные курсоры (Explicit Cursors):

- Объявляются и управляются явно разработчиком.

- Используются для обработки набора строк, возвращаемого SELECT-запросом.

2. Неявные курсоры (Implicit Cursors):

- Создаются и управляются автоматически Oracle для каждого оператора DML (INSERT, UPDATE, DELETE) и операторов SELECT INTO.

- Не требуют явного объявления.

Схемы обработки курсора

1. Объявление курсора:

CURSOR cursor\_name IS

SELECT column1, column2 FROM table\_name WHERE condition;

2. Открытие курсора:

OPEN cursor\_name;

3. Извлечение данных (FETCH) из курсора:

FETCH cursor\_name INTO variable1, variable2;

4. Закрытие курсора:

CLOSE cursor\_name;

Атрибуты курсора

Курсоры в PL/SQL обладают атрибутами, которые помогают управлять ими и получать информацию о состоянии курсора:

1. %FOUND:

- Возвращает TRUE, если последняя операция FETCH была успешной.

2. %NOTFOUND:

- Возвращает TRUE, если последняя операция FETCH не вернула строку.

3. %ISOPEN:

- Возвращает TRUE, если курсор открыт.

4. %ROWCOUNT:

- Возвращает количество строк, извлеченных из курсора до текущего момента.

Курсоры с параметрами

Курсоры с параметрами позволяют передавать значения в курсор для выполнения запроса с параметрами.

Динамические курсоры

Динамические курсоры используются для выполнения SQL-запросов, которые формируются динамически во время выполнения программы. Для работы с динамическими курсорами используются пакеты “DBMS\_SQL”.

1. Курсоры. Курсорные переменные. Курсорные подзапросы. Использование конструкции CURRENT OF в курсорах.

Курсоры в PL/SQL позволяют разработчику обрабатывать множество строк, полученных от SQL-запроса. Есть два основных типа курсоров: неявные и явные.

1. Явные курсоры (Explicit Cursors):

- Объявляются и управляются явно.

- Используются для обработки набора строк, возвращенного SELECT-запросом.

2. Неявные курсоры (Implicit Cursors):

- Создаются и управляются автоматически для каждого оператора DML (INSERT, UPDATE, DELETE) и оператора SELECT INTO.

- Не требуют явного объявления.

Курсорные переменные (Ref Cursors)

Курсорные переменные, также известные как REF CURSORs, предоставляют гибкость в управлении и передаче курсоров между различными блоками PL/SQL.

Курсорные подзапросы

Курсорные подзапросы позволяют использовать курсор внутри SQL-запроса.

Использование конструкции “CURRENT OF” в курсорах

Конструкция “CURRENT OF” используется для обновления или удаления текущей строки, на которую указывает курсор.

1. Обработка исключений в PL/SQL, стандартные исключения, генерация и обработка исключения.

Обработка исключений в PL/SQL

Обработка исключений включает три основных шага:

1. Выявление исключения: Код, который может вызвать исключение, помещается в блок “BEGIN”.

2. Определение исключения: Исключение обрабатывается в блоке “EXCEPTION”.

3. Обработка исключения: Код, который выполняется в случае возникновения исключения, находится в блоке “EXCEPTION”.

Стандартные исключения

PL/SQL предоставляет несколько встроенных стандартных исключений, которые охватывают распространенные ошибки. Вот некоторые из них:

1. NO\_DATA\_FOUND: Возникает, когда SELECT INTO не возвращает строки.

2. TOO\_MANY\_ROWS: Возникает, когда SELECT INTO возвращает более одной строки.

3. ZERO\_DIVIDE: Возникает при попытке деления на ноль.

4. INVALID\_CURSOR: Возникает при неправильной работе с курсором.

5. DUP\_VAL\_ON\_INDEX: Возникает при попытке вставить дублирующееся значение в уникальный индекс.

Генерация пользовательских исключений

PL/SQL позволяет разработчикам создавать пользовательские исключения и генерировать их с помощью оператора “RAISE”.

1. Принцип распространения исключений в PL/SQL. Инструкция RAISE\_APPLICATION\_ERROR.

В PL/SQL исключения могут распространяться ("propagate") по стеку вызовов от внутреннего блока к внешнему. Если исключение не обработано в текущем блоке, оно передается на уровень выше до тех пор, пока не будет обработано или не достигнет самого внешнего блока.

Принцип распространения исключений в PL/SQL

Когда исключение возникает в блоке PL/SQL, происходит следующее:

1. PL/SQL сначала ищет блок обработки исключений внутри текущего блока.

2. Если подходящий блок обработки исключений найден, исключение обрабатывается.

3. Если обработчик исключений не найден, исключение распространяется на внешний блок (если такой существует).

4. Этот процесс продолжается до тех пор, пока не будет найден обработчик исключений или не будет достигнут самый внешний блок.

5. Если исключение не обработано и достигает самого внешнего блока, выполнение программы завершится с ошибкой.

Инструкция RAISE\_APPLICATION\_ERROR

Инструкция “RAISE\_APPLICATION\_ERROR” используется для генерации пользовательских ошибок в программах PL/SQL. Она позволяет разработчикам создавать собственные сообщения об ошибках и возвращать их в вызывающее приложение.

Синтаксис “RAISE\_APPLICATION\_ERROR”:

RAISE\_APPLICATION\_ERROR(error\_number, message[, {TRUE | FALSE}]);

- “error\_number”: Номер ошибки, который должен быть в диапазоне от -20000 до -20999.

- “message”: Сообщение об ошибке, которое будет возвращено.

- Третий параметр (опциональный): TRUE или FALSE. Если TRUE, исключение включается в стек вызовов; по умолчанию FALSE.

Преимущества использования “RAISE\_APPLICATION\_ERROR”

1. Улучшение управления ошибками: Позволяет разработчикам создавать и обрабатывать специфические для приложения ошибки.

2. Читаемость кода: Улучшает читаемость и поддержку кода, предоставляя понятные сообщения об ошибках.

3. Инкапсуляция логики ошибок: Обеспечивает более надежное управление ошибками, позволяя точно определить, когда и почему возникает ошибка.

1. Встроенные функции языка PL/SQL. Функции работы с датами, текстом и числами.

В языке PL/SQL Oracle предоставляет множество встроенных функций для работы с датами, текстом и числами. Эти функции облегчают выполнение различных операций и обработку данных.

Функции работы с датами

1. SYSDATE:

- Возвращает текущую дату и время сервера.

2. ADD\_MONTHS:

- Добавляет указанное количество месяцев к дате.

3. MONTHS\_BETWEEN:

- Возвращает разницу в месяцах между двумя датами.

4. NEXT\_DAY:

- Возвращает дату следующего указанного дня недели после заданной даты.

5. TRUNC (Date):

- Усекание даты до указанного уровня (день, месяц, год и т.д.).

Функции работы с текстом

1. SUBSTR:

- Возвращает подстроку из строки.

2. INSTR:

- Возвращает позицию подстроки в строке.

3. LENGTH:

- Возвращает длину строки.

4. UPPER:

- Преобразует строку в верхний регистр.

5. LOWER:

- Преобразует строку в нижний регистр.

Функции работы с числами

1. ROUND (Number):

- Округляет число до указанного количества знаков.

2. TRUNC (Number):

- Усекание числа до указанного количества знаков.

3. MOD:

- Возвращает остаток от деления двух чисел.

4. CEIL:

- Возвращает наименьшее целое число, большее или равное указанному числу.

5. FLOOR:

- Возвращает наибольшее целое число, меньшее или равное указанному числу.

1. Коллекции. Массивы переменной длины. Вложенные таблицы. Ассоциативные массивы.

Коллекции в PL/SQL Oracle представляют собой структуру данных, которая позволяет хранить и управлять множеством элементов одного типа. Основными типами коллекций являются массивы переменной длины (VARRAY), вложенные таблицы и ассоциативные массивы.

Массивы переменной длины (VARRAY)

Массивы переменной длины, или VARRAY, позволяют хранить фиксированное количество элементов, определенное при создании массива. Они используются, когда количество элементов заранее известно и не изменяется.

Вложенные таблицы

Вложенные таблицы представляют собой коллекцию, которая может динамически изменять количество элементов. В отличие от VARRAY, вложенные таблицы могут содержать неограниченное количество элементов.

Ассоциативные массивы

Ассоциативные массивы (ранее известные как индексируемые по строке таблицы) представляют собой коллекцию пар "ключ-значение". Они позволяют обращаться к элементам массива по ключу, который может быть строкой или числом.

Пример с использованием всех типов коллекций

DECLARE

-- Объявление типа VARRAY

TYPE varray\_type IS VARRAY(3) OF NUMBER;

v\_numbers varray\_type := varray\_type(1, 2, 3);

-- Объявление типа вложенной таблицы

TYPE nested\_table\_type IS TABLE OF VARCHAR2(50);

v\_names nested\_table\_type := nested\_table\_type();

-- Объявление типа ассоциативного массива

TYPE associative\_array\_type IS TABLE OF VARCHAR2(50) INDEX BY PLS\_INTEGER;

v\_colors associative\_array\_type;

BEGIN

-- Добавление элементов в вложенную таблицу

v\_names.EXTEND; v\_names(1) := 'Alice';

v\_names.EXTEND; v\_names(2) := 'Bob';

v\_names.EXTEND; v\_names(3) := 'Charlie';

-- Добавление элементов в ассоциативный массив

v\_colors(1) := 'Red';

v\_colors(2) := 'Green';

v\_colors(3) := 'Blue';

-- Вывод элементов всех коллекций

FOR i IN 1..v\_numbers.COUNT LOOP

DBMS\_OUTPUT.PUT\_LINE('VARRAY Element ' || i || ': ' || v\_numbers(i));

END LOOP;

FOR i IN 1..v\_names.COUNT LOOP

DBMS\_OUTPUT.PUT\_LINE('Nested Table Name ' || i || ': ' || v\_names(i));

END LOOP;

FOR i IN 1..v\_colors.COUNT LOOP

DBMS\_OUTPUT.PUT\_LINE('Associative Array Color ' || i || ': ' || v\_colors(i));

END LOOP;

END;

/

1. Процедурные объекты. Хранимые процедуры. Вызов процедур. Входные и выходные параметры, позиционный и параметрический форматы передачи фактических параметров. Значения параметров по умолчанию.

Процедурные объекты

Процедурные объекты в PL/SQL — это хранимые процедуры и функции, которые позволяют выполнять повторяющиеся задачи, организовывать код и улучшать его модульность и читаемость. Они хранятся в базе данных и могут быть вызваны из различных программ или скриптов.

Хранимые процедуры

Хранимая процедура — это именованный блок PL/SQL, который выполняет определенные действия и может принимать параметры. Хранимые процедуры позволяют улучшить производительность и безопасность приложений, так как код выполняется на стороне сервера.

Вызов процедур

Хранимые процедуры могут быть вызваны из PL/SQL-блоков, триггеров, других процедур или пакетов.

Входные и выходные параметры

Параметры процедуры могут быть входными (“IN”), выходными (“OUT”) или входными/выходными (“IN OUT”).

Позиционный и параметрический форматы передачи фактических параметров

Параметры могут передаваться в процедуру в позиционном или параметрическом формате.

1. Позиционный формат:

- Параметры передаются в порядке их объявления.

2. Параметрический формат:

- Параметры передаются с указанием имен.

Значения параметров по умолчанию

Параметры процедур могут иметь значения по умолчанию, которые используются, если фактические параметры не передаются.

Пример использования всех концепций

Создание процедуры:

CREATE OR REPLACE PROCEDURE calculate\_salary (

p\_base\_salary IN NUMBER,

p\_bonus OUT NUMBER,

p\_tax\_rate IN NUMBER DEFAULT 0.20

) AS

BEGIN

p\_bonus := p\_base\_salary \* 0.10;

DBMS\_OUTPUT.PUT\_LINE('Base Salary: ' || p\_base\_salary);

DBMS\_OUTPUT.PUT\_LINE('Bonus: ' || p\_bonus);

DBMS\_OUTPUT.PUT\_LINE('Tax Rate: ' || p\_tax\_rate);

END calculate\_salary;

/

“““

Вызов процедуры:

“““sql

DECLARE

v\_base\_salary NUMBER := 5000;

v\_bonus NUMBER;

BEGIN

-- Позиционный вызов

calculate\_salary(v\_base\_salary, v\_bonus);

-- Параметрический вызов

calculate\_salary(p\_base\_salary => v\_base\_salary, p\_bonus => v\_bonus, p\_tax\_rate => 0.25);

END;

/

1. Процедурные объекты. Хранимые функции. Параметры функции. Вызов функций. Понятие детерминированной функции. Понятие pipeline функции. Значения параметров по умолчанию.

Хранимые функции

Хранимая функция — это именованный блок PL/SQL, который выполняет определенные действия и возвращает значение. Функции используются для вычисления и возвращения значений, и они могут принимать параметры.

Пример создания хранимой функции:

“““sql

CREATE OR REPLACE FUNCTION add\_numbers (

p\_num1 IN NUMBER,

p\_num2 IN NUMBER

) RETURN NUMBER AS

BEGIN

RETURN p\_num1 + p\_num2;

END add\_numbers;

/

Параметры функции

Параметры функции могут быть входными (“IN”), но не могут быть выходными (“OUT”) или входными/выходными (“IN OUT”), так как функция должна возвращать только одно значение.

Пример функции с параметрами:

“““sql

CREATE OR REPLACE FUNCTION get\_full\_name (

p\_first\_name IN VARCHAR2,

p\_last\_name IN VARCHAR2

) RETURN VARCHAR2 AS

BEGIN

RETURN p\_first\_name || ' ' || p\_last\_name;

END get\_full\_name;

/

Вызов функций

Функции могут быть вызваны из PL/SQL-блоков, других функций, процедур или пакетов, а также в SQL-запросах.

Пример вызова функции из PL/SQL блока:

“““sql

DECLARE

v\_full\_name VARCHAR2(100);

BEGIN

v\_full\_name := get\_full\_name('John', 'Doe');

DBMS\_OUTPUT.PUT\_LINE('Full Name: ' || v\_full\_name);

END;

/

Пример вызова функции в SQL-запросе:

“““sql

SELECT get\_full\_name(first\_name, last\_name) AS full\_name

FROM employees

WHERE employee\_id = 101;

Детерминированная функция

Детерминированная функция — это функция, которая всегда возвращает одно и то же значение при одинаковых входных параметрах. Эти функции могут быть помечены ключевым словом “DETERMINISTIC”, что позволяет Oracle кэшировать результаты их вызовов и улучшать производительность.

Пример детерминированной функции:

“““sql

CREATE OR REPLACE FUNCTION square\_number (

p\_num IN NUMBER

) RETURN NUMBER DETERMINISTIC AS

BEGIN

RETURN p\_num \* p\_num;

END square\_number;

/

Pipeline функции

Pipeline функции — это функции, которые возвращают набор данных по одной строке за раз. Они используются для обработки больших наборов данных и позволяют сократить использование памяти и улучшить производительность.

Пример pipeline функции:

Создание типа таблицы:

“““sql

CREATE OR REPLACE TYPE number\_table AS TABLE OF NUMBER;

/

Создание pipeline функции:

“““sql

CREATE OR REPLACE FUNCTION generate\_numbers (

p\_max\_number IN NUMBER

) RETURN number\_table PIPELINED AS

BEGIN

FOR i IN 1..p\_max\_number LOOP

PIPE ROW(i);

END LOOP;

RETURN;

END generate\_numbers;

/

Вызов pipeline функции:

“““sql

SELECT \* FROM TABLE(generate\_numbers(10));

Значения параметров по умолчанию

Функции могут иметь значения параметров по умолчанию, которые используются, если фактические параметры не передаются при вызове функции.

Пример функции со значением параметра по умолчанию:

“““sql

CREATE OR REPLACE FUNCTION greet (

p\_name IN VARCHAR2 DEFAULT 'Guest'

) RETURN VARCHAR2 AS

BEGIN

RETURN 'Hello, ' || p\_name || '!';

END greet;

/

Пример вызова функции со значением параметра по умолчанию:

“““sql

DECLARE

v\_greeting VARCHAR2(100);

BEGIN

v\_greeting := greet; -- Используется значение по умолчанию 'Guest'

DBMS\_OUTPUT.PUT\_LINE(v\_greeting);

v\_greeting := greet('Alice'); -- Передается фактический параметр

DBMS\_OUTPUT.PUT\_LINE(v\_greeting);

END;

/

1. Процедурные объекты. Пакеты. Спецификация и реализация пакета.

Пакеты в PL/SQL Oracle представляют собой коллекцию связанных процедур, функций, курсоров и переменных. Они позволяют организовать и структурировать код, делая его более модульным и удобным для сопровождения. Пакеты состоят из двух частей: спецификации и тела (реализации).

Спецификация пакета

Спецификация пакета (package specification) определяет интерфейс пакета, который включает объявления всех публичных элементов (процедур, функций, курсоров и переменных). Спецификация видна всем пользователям базы данных, которые имеют доступ к пакету.

Пример спецификации пакета:

“““sql

CREATE OR REPLACE PACKAGE math\_pkg AS

-- Объявление функции

FUNCTION add\_numbers (p\_num1 IN NUMBER, p\_num2 IN NUMBER) RETURN NUMBER;

-- Объявление процедуры

PROCEDURE display\_sum (p\_num1 IN NUMBER, p\_num2 IN NUMBER);

-- Объявление переменной

pi CONSTANT NUMBER := 3.14159;

END math\_pkg;

/

Тело (реализация) пакета

Тело пакета (package body) содержит реализацию всех процедур, функций и других элементов, объявленных в спецификации. Тело пакета может также содержать приватные элементы, которые не видны за пределами пакета.

Пример тела пакета:

“““sql

CREATE OR REPLACE PACKAGE BODY math\_pkg AS

-- Реализация функции

FUNCTION add\_numbers (p\_num1 IN NUMBER, p\_num2 IN NUMBER) RETURN NUMBER AS

BEGIN

RETURN p\_num1 + p\_num2;

END add\_numbers;

-- Реализация процедуры

PROCEDURE display\_sum (p\_num1 IN NUMBER, p\_num2 IN NUMBER) AS

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Sum: ' || add\_numbers(p\_num1, p\_num2));

END display\_sum;

END math\_pkg;

/

Вызов элементов пакета

Элементы пакета могут быть вызваны из любого PL/SQL блока, триггера, процедуры или функции, если у пользователя есть соответствующие права доступа.

Пример вызова элементов пакета:

“““sql

BEGIN

-- Вызов функции из пакета

DBMS\_OUTPUT.PUT\_LINE('Sum: ' || math\_pkg.add\_numbers(10, 20));

-- Вызов процедуры из пакета

math\_pkg.display\_sum(15, 25);

END;

/

1. Процедурные объекты. Триггеры. Виды триггеров. Классификация, порядок выполнения и предикаты триггеров. Триггеры замещения. Привилегии для создания триггеров. Включение/отключение триггеров. Псевдозаписи old и new.

Триггеры в Oracle — это специальные типы хранимых процедур, которые автоматически выполняются в ответ на определенные события в базе данных, такие как вставка, обновление или удаление строк в таблице. Давайте рассмотрим различные аспекты триггеров, включая их виды, классификацию, порядок выполнения, предикаты, триггеры замещения, привилегии для создания триггеров, их включение/отключение и использование псевдозаписей “OLD” и “NEW”.

Виды триггеров

1. DML триггеры (Data Manipulation Language):

- Вставка (“INSERT”): Срабатывают перед или после вставки строки в таблицу.

- Обновление (“UPDATE”): Срабатывают перед или после обновления строки в таблице.

- Удаление (“DELETE”): Срабатывают перед или после удаления строки из таблицы.

2. DDL триггеры (Data Definition Language):

- Срабатывают при выполнении команд DDL, таких как “CREATE”, “ALTER”, “DROP”.

- Пример: Журналирование изменений в структуре базы данных.

3. Триггеры базы данных:

- Срабатывают при событиях на уровне базы данных, таких как вход/выход пользователя, старт или остановка базы данных.

Классификация триггеров

1. По времени срабатывания:

- BEFORE: Срабатывают перед выполнением основной операции.

- AFTER: Срабатывают после выполнения основной операции.

- INSTEAD OF: Замещающие триггеры для представлений, выполняются вместо основной операции.

2. По количеству строк:

- ROW: Выполняются для каждой строки, затронутой операцией.

- STATEMENT: Выполняются один раз для всей операции, независимо от количества затронутых строк.

Порядок выполнения триггеров

Порядок выполнения триггеров определяется следующими правилами:

1. BEFORE триггеры: Срабатывают перед основной операцией (вставка, обновление, удаление).

2. Операция SQL: Выполняется основная операция.

3. AFTER триггеры: Срабатывают после основной операции.

Предикаты триггеров

Предикаты триггеров используются для определения условий, при которых триггер должен сработать. Это позволяет контролировать, когда и при каких условиях триггер выполняется.

Пример использования предиката триггера:

“““sql

CREATE OR REPLACE TRIGGER trg\_update\_salary

BEFORE UPDATE OF salary ON employees

FOR EACH ROW

WHEN (NEW.salary > 5000)

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Salary is being updated to a value greater than 5000');

END trg\_update\_salary;

/

Триггеры замещения (INSTEAD OF)

Триггеры замещения используются для представлений и выполняются вместо основной операции (вставка, обновление, удаление).

Пример триггера замещения:

“““sql

CREATE OR REPLACE TRIGGER trg\_instead\_of\_insert

INSTEAD OF INSERT ON employee\_view

FOR EACH ROW

BEGIN

INSERT INTO employees (employee\_id, first\_name, last\_name)

VALUES (:NEW.employee\_id, :NEW.first\_name, :NEW.last\_name);

END trg\_instead\_of\_insert;

/

Привилегии для создания триггеров

Для создания триггеров пользователь должен обладать следующими привилегиями:

1. Привилегия “CREATE TRIGGER”: Для создания триггеров на уровне пользователя.

2. Привилегия “ALTER” на таблицу/представление: Для создания триггеров на уровне таблицы или представления.

Включение/отключение триггеров

Триггеры могут быть временно отключены или снова включены с использованием команд “ALTER TRIGGER” и “ALTER TABLE”.

Пример включения/отключения триггера:

“““sql

-- Отключение триггера

ALTER TRIGGER trg\_update\_salary DISABLE;

-- Включение триггера

ALTER TRIGGER trg\_update\_salary ENABLE;

Псевдозаписи “OLD” и “NEW”

Псевдозаписи “OLD” и “NEW” используются внутри триггеров для доступа к старым и новым значениям столбцов строк, затронутых операцией.

Пример использования псевдозаписей “OLD” и “NEW”:

“““sql

CREATE OR REPLACE TRIGGER trg\_audit\_changes

BEFORE UPDATE ON employees

FOR EACH ROW

BEGIN

INSERT INTO audit\_table (employee\_id, old\_salary, new\_salary, change\_date)

VALUES (:OLD.employee\_id, :OLD.salary, :NEW.salary, SYSDATE);

END trg\_audit\_changes;

/

1. Секционирование таблиц. Виды секционирования.

Секционирование таблиц в Oracle — это механизм, позволяющий разделить большую таблицу на более мелкие, управляемые сегменты, называемые секциями (partitions). Это улучшает производительность запросов и управление данными, поскольку операции могут быть выполнены на отдельных секциях, а не на всей таблице. Давайте рассмотрим основные виды секционирования таблиц в Oracle.

Виды секционирования

1. Секционирование по диапазону (Range Partitioning):

- Деление данных на секции на основе диапазона значений.

- Пример: Секционирование таблицы по дате.

“““sql

CREATE TABLE sales (

sale\_id NUMBER,

sale\_date DATE,

amount NUMBER

)

PARTITION BY RANGE (sale\_date) (

PARTITION sales\_2021\_q1 VALUES LESS THAN (TO\_DATE('01-APR-2021', 'DD-MON-YYYY')),

PARTITION sales\_2021\_q2 VALUES LESS THAN (TO\_DATE('01-JUL-2021', 'DD-MON-YYYY')),

PARTITION sales\_2021\_q3 VALUES LESS THAN (TO\_DATE('01-OCT-2021', 'DD-MON-YYYY')),

PARTITION sales\_2021\_q4 VALUES LESS THAN (TO\_DATE('01-JAN-2022', 'DD-MON-YYYY'))

);

2. Секционирование по списку (List Partitioning):

- Деление данных на секции на основе списка значений.

- Пример: Секционирование таблицы по региону.

“““sql

CREATE TABLE customers (

customer\_id NUMBER,

customer\_name VARCHAR2(100),

region VARCHAR2(50)

)

PARTITION BY LIST (region) (

PARTITION east VALUES ('East'),

PARTITION west VALUES ('West'),

PARTITION north VALUES ('North'),

PARTITION south VALUES ('South')

);

3. Секционирование по хэшу (Hash Partitioning):

- Деление данных на секции на основе хэш-функции, что обеспечивает равномерное распределение данных.

- Пример: Секционирование таблицы по идентификатору клиента.

“““sql

CREATE TABLE transactions (

transaction\_id NUMBER,

customer\_id NUMBER,

amount NUMBER

)

PARTITION BY HASH (customer\_id) (

PARTITIONS 4

);

4. Секционирование по составному критерию (Composite Partitioning):

- Комбинация двух или более методов секционирования, таких как секционирование по диапазону и хэшу.

- Пример: Секционирование таблицы по дате и клиенту.

“““sql

CREATE TABLE orders (

order\_id NUMBER,

order\_date DATE,

customer\_id NUMBER,

amount NUMBER

)

PARTITION BY RANGE (order\_date)

SUBPARTITION BY HASH (customer\_id)

SUBPARTITIONS 4 (

PARTITION orders\_2021\_q1 VALUES LESS THAN (TO\_DATE('01-APR-2021', 'DD-MON-YYYY')),

PARTITION orders\_2021\_q2 VALUES LESS THAN (TO\_DATE('01-JUL-2021', 'DD-MON-YYYY')),

PARTITION orders\_2021\_q3 VALUES LESS THAN (TO\_DATE('01-OCT-2021', 'DD-MON-YYYY')),

PARTITION orders\_2021\_q4 VALUES LESS THAN (TO\_DATE('01-JAN-2022', 'DD-MON-YYYY'))

);

5. Секционирование по ключу (Key Partitioning):

- Разновидность хэш-секционирования, в которой используются уникальные ключи таблицы.

- Пример: Секционирование таблицы по идентификатору заказа.

“““sql

CREATE TABLE shipments (

shipment\_id NUMBER,

order\_id NUMBER,

shipment\_date DATE,

status VARCHAR2(50)

)

PARTITION BY HASH (order\_id) (

PARTITIONS 8

);

Преимущества секционирования таблиц

1. Улучшение производительности: Улучшается производительность запросов, так как они могут выполняться на отдельных секциях таблицы.

2. Управляемость: Облегчает управление данными, так как операции могут выполняться на уровне секции.

3. Архивирование и очистка данных: Упрощается архивирование и очистка данных, так как можно работать с отдельными секциями.

1. Транзакции в СУБД Oracle. Виды транзакций. Понятие автономной транзакции.

Транзакции в СУБД Oracle — это группы операций SQL, которые выполняются как единое целое, обеспечивая целостность и согласованность данных. Транзакции обладают четырьмя основными свойствами, известными как ACID (Atomicity, Consistency, Isolation, Durability).

Основные свойства транзакций (ACID)

1. Атомарность (Atomicity): Транзакция выполняется целиком или не выполняется вовсе. Если одна из операций не может быть выполнена, все изменения откатываются.

2. Согласованность (Consistency): После выполнения транзакции база данных остается в согласованном состоянии.

3. Изоляция (Isolation): Транзакции выполняются независимо друг от друга, не влияя друг на друга.

4. Долговечность (Durability): После завершения транзакции изменения сохраняются в базе данных и не могут быть потеряны.

Виды транзакций

В Oracle можно выделить несколько типов транзакций, включая стандартные транзакции и автономные транзакции.

1. Стандартные транзакции:

- Начинаются автоматически при выполнении первой команды DML (INSERT, UPDATE, DELETE).

- Завершаются явным выполнением команды “COMMIT” или “ROLLBACK”.

Пример стандартной транзакции:

“““sql

BEGIN

-- Начало транзакции

INSERT INTO employees (employee\_id, first\_name, last\_name)

VALUES (101, 'John', 'Doe');

-- Завершение транзакции с фиксацией изменений

COMMIT;

END;

2. Автономные транзакции:

- Являются независимыми от основной транзакции и могут выполняться внутри другой транзакции.

- Полезны для операций, которые должны быть зафиксированы независимо от основной транзакции (например, запись журнала аудита).

Понятие автономной транзакции

Автономная транзакция — это специальный вид транзакции, которая выполняется независимо от основной транзакции. Она может фиксировать (“COMMIT”) или откатывать (“ROLLBACK”) свои изменения без влияния на состояние основной транзакции.

Пример автономной транзакции:

“““sql

CREATE OR REPLACE PROCEDURE log\_transaction (

p\_message IN VARCHAR2

) IS

PRAGMA AUTONOMOUS\_TRANSACTION;

BEGIN

-- Вставка записи в журнал аудита

INSERT INTO audit\_log (log\_message, log\_date)

VALUES (p\_message, SYSDATE);

-- Фиксация изменений в автономной транзакции

COMMIT;

END log\_transaction;

/

BEGIN

-- Начало основной транзакции

UPDATE employees

SET salary = salary \* 1.1

WHERE department\_id = 10;

-- Вызов автономной транзакции для записи в журнал аудита

log\_transaction('Salaries updated for department 10');

-- Завершение основной транзакции с фиксацией изменений

COMMIT;

END;

В этом примере процедура “log\_transaction” выполняется как автономная транзакция благодаря использованию директивы “PRAGMA AUTONOMOUS\_TRANSACTION”. Она фиксирует запись в журнале аудита независимо от основной транзакции.

1. Обработка заданий. Системные пакеты обработки заданий в Oracle.

Обработка заданий в Oracle — это процесс автоматизации выполнения задач и операций в базе данных по расписанию или в ответ на определенные события. Oracle предоставляет мощные системные пакеты для обработки заданий, такие как DBMS\_SCHEDULER и DBMS\_JOB. Давайте рассмотрим эти пакеты подробнее.

DBMS\_SCHEDULER

Пакет DBMS\_SCHEDULER предоставляет гибкие средства для планирования и управления заданиями. С его помощью можно создавать, изменять, удалять и мониторить задания, а также настраивать сложные расписания и зависимости между заданиями.

Основные процедуры DBMS\_SCHEDULER:

1. CREATE\_JOB:

- Создание нового задания.

“““sql

BEGIN

DBMS\_SCHEDULER.CREATE\_JOB (

job\_name => 'my\_job',

job\_type => 'PLSQL\_BLOCK',

job\_action => 'BEGIN DBMS\_OUTPUT.PUT\_LINE(''Hello, World!''); END;',

start\_date => SYSTIMESTAMP,

repeat\_interval => 'FREQ=DAILY; BYHOUR=12; BYMINUTE=0; BYSECOND=0',

enabled => TRUE

);

END;

/

2. ENABLE:

- Включение задания.

“““sql

BEGIN

DBMS\_SCHEDULER.ENABLE('my\_job');

END;

/

3. DISABLE:

- Отключение задания.

“““sql

BEGIN

DBMS\_SCHEDULER.DISABLE('my\_job');

END;

/

4. DROP\_JOB:

- Удаление задания.

“““sql

BEGIN

DBMS\_SCHEDULER.DROP\_JOB('my\_job');

END;

/

5. SET\_ATTRIBUTE:

- Изменение атрибута задания.

“““sql

BEGIN

DBMS\_SCHEDULER.SET\_ATTRIBUTE('my\_job', 'repeat\_interval', 'FREQ=HOURLY; INTERVAL=2');

END;

6. RUN\_JOB:

- Запуск задания немедленно.

“““sql

BEGIN

DBMS\_SCHEDULER.RUN\_JOB('my\_job');

END;

/

DBMS\_JOB

Пакет DBMS\_JOB предоставляет основные средства для создания и управления заданиями. Он является предшественником DBMS\_SCHEDULER и обеспечивает базовую функциональность для планирования заданий.

Основные процедуры DBMS\_JOB:

1. SUBMIT:

- Создание нового задания.

“““sql

DECLARE

job\_number NUMBER;

BEGIN

DBMS\_JOB.SUBMIT(

job => job\_number,

what => 'BEGIN DBMS\_OUTPUT.PUT\_LINE(''Hello, World!''); END;',

next\_date => SYSDATE,

interval => 'SYSDATE + 1'

);

COMMIT;

END;

/

2. REMOVE:

- Удаление задания.

“““sql

BEGIN

DBMS\_JOB.REMOVE(job\_number);

COMMIT;

END;

/

3. CHANGE:

- Изменение следующей даты выполнения и интервала задания.

“““sql

BEGIN

DBMS\_JOB.CHANGE(

job => job\_number,

what => 'BEGIN DBMS\_OUTPUT.PUT\_LINE(''Updated Job''); END;',

next\_date => SYSDATE,

interval => 'SYSDATE + 2'

);

COMMIT;

END;

/

4. RUN:

- Запуск задания немедленно.

“““sql

BEGIN

DBMS\_JOB.RUN(job\_number);

END;

/

Преимущества DBMS\_SCHEDULER по сравнению с DBMS\_JOB

1. Более гибкие расписания: DBMS\_SCHEDULER предоставляет возможности для создания сложных расписаний с использованием выражений календарных интервалов.

2. Управление ресурсами: Позволяет управлять приоритетами и ограничивать использование ресурсов для заданий.

3. Отслеживание и мониторинг: Включает в себя встроенные средства для отслеживания состояния заданий и записи журналов выполнения.