

## 1. SQL, SQL/PSM

**1. История и этапы развития SQL. Организации, принимающие стандарты. Уровни соответствия, расширения и диалекты. Классификация интерфейсов SQL. Типы данных SQL: простые, коллекции, LOB и проблемы их использования, приоритеты типов. Особенности типов данных: логического, даты и времени, пользовательского, UDT в Oracle/SQL Server. Объектные типы в Oracle.**

SQL (Structured Query Language) — декларативный язык программирования, предназначенный для управления реляционными базами данных.

Основные этапы развития:

- 1970-е: Зарождение (IBM).
- 1980-е: Стандартизация и коммерческое внедрение.
- 1990-е: Расширение функциональности.
- 2000-е: Специализация и XML.
- 2010-е: Big Data и JSON.

Организации, принимающие стандарты:

- ANSI (American National Standards Institute) — американский институт стандартов, первичный разработчик стандартов SQL.
- ISO (International Organization for Standardization) — международная организация, выпускающая стандарты ISO/IEC 9075 (SQL).

Уровни соответствия стандарту:

- Entry Level — базовый уровень (устаревший после SQL-92).
- Core SQL — полная поддержка основных возможностей стандарта.
- Enhanced SQL — поддержка расширенных функций (оконные функции, XML, JSON).

Расширения SQL:

- SQL/PSM — процедурные расширения (циклы, условия).
- SQL/CLI — интерфейс для доступа через вызовы функций (ODBC, JDBC).
- SQL/OLAP — оперативно-аналитическая обработка.
- SQL/XML — интеграция XML.

Диалекты SQL:

- PL/SQL (Oracle) — процедурный язык с интеграцией SQL.
- T-SQL — расширенный синтаксис, процедурные конструкции.
- PL/pgSQL (PostgreSQL) — аналог PL/SQL для PostgreSQL.
- SQL PL (IBM DB2) — процедурный язык DB2.

Классификация интерфейсов SQL:

- Интерактивный SQL
- Встраиваемый SQL (Embedded SQL)
- Модульный SQL
- Интерфейсы уровня вызовов (Call-Level Interfaces, CLI)
- Веб-интерфейсы и ORM

Типы данных SQL

Простые (скалярные) типы данных:

- Числовые: INT, SMALLINT, DECIMAL(p,s), FLOAT, REAL.
- Строковые: CHAR(n), VARCHAR(n), CLOB (символьные большие объекты).
- Бинарные: BINARY, VARBINARY, BLOB (бинарные большие объекты).
- Дата и время: DATE, TIME, TIMESTAMP, INTERVAL.
- Логический тип: BOOLEAN (TRUE, FALSE, UNKNOWN).

#### Типы-коллекции

- Массивы (ARRAY) — упорядоченные наборы элементов одного типа.
- Множества (MULTISET) — неупорядоченные коллекции с возможностью дубликатов.

#### LOB (Large Objects):

- CLOB (Character LOB) — для больших текстов (до 4 ГБ и более).
- BLOB (Binary LOB) — для бинарных данных (изображения, видео).
- NCLOB — для Unicode-текстов.

#### Проблемы использования LOB:

- Высокое потребление ресурсов
- Ограниченная индексация
- Транзакционные сложности
- Неэффективность в операциях соединения и сортировки — LOB нельзя использовать в GROUP BY, ORDER BY без преобразований

#### Приоритеты типов данных:

При выполнении операций между разными типами данных происходит неявное преобразование согласно приоритетам.

#### Особенности отдельных типов данных

##### Логический тип (BOOLEAN):

- В стандарте SQL:1999. Принимает TRUE, FALSE, UNKNOWN
- Oracle: до версии 23c не имел native BOOLEAN в SQL (только в PL/SQL)
- SQL Server: нет native BOOLEAN, используется BIT (0/1)

#### Типы даты и времени

- DATE — дата (год, месяц, день).
- TIME — время с точностью до долей секунды.
- TIMESTAMP — дата и время с временной зоной или без неё.
- INTERVAL — интервалы времени (год-месяц или день-секунда).

#### Особенности:

- Oracle: DATE включает время (с точностью до секунды). Есть TIMESTAMP(9) с наносекундной точностью.
- SQL Server: DATETIME (точность ~3.33 мс), DATETIME2 (высокая точность), DATETIMEOFFSET (с временной зоной).

#### Пользовательские типы данных (UDT — User-Defined Types)

DOMAIN — ограничение на существующий тип (например, CREATE DOMAIN Email AS VARCHAR(255) CHECK ...).

DISTINCT TYPE — логически отдельный тип на основе существующего (например, CREATE TYPE Money AS DECIMAL(10,2)).

UDT в Oracle и SQL Server

Oracle:

Object Types: создаются через CREATE TYPE ... AS OBJECT. Могут содержать атрибуты и методы.

```
CREATE TYPE Address AS OBJECT (  
    street VARCHAR2(100),  
    city VARCHAR2(50)  
);
```

SQL Server:

Alias Types: CREATE TYPE PhoneNumber FROM VARCHAR(20) NOT NULL.

CLR UDT (на основе .NET): создаются на языках C#, VB.NET, компилируются в сборки.

Объектные типы в Oracle

Oracle поддерживает объектно-ориентированную парадигму через:

- Объектные таблицы — таблицы, хранящие объекты. CREATE TABLE employees OF PersonType;
- Наследование типов:  
CREATE TYPE EmployeeType UNDER PersonType (...);
- Методы в типах:  
CREATE TYPE BankAccount AS OBJECT (  
 balance NUMBER,  
 MEMBER FUNCTION withdraw(amount NUMBER) RETURN NUMBER  
);
- Референции (REF) — указатели на объекты в объектных таблицах.

**2. Идентификаторы. Наборы символов: назначение, определение, сопоставление, отмена, установка. Домены: назначение, создание, изменение, удаление, особенности ограничений, значений по умолчанию, правила сравнений символьных типов. Утверждения: назначение, создание, удаление, отложенные/не отложенные. Столбцы типа XML в Oracle/DB2/SQL Server.**

Назначение: Идентификаторы используются для именования объектов базы данных. Они обеспечивают уникальность и ссылочную целостность.

Определение: Идентификатор — это последовательность символов, задаваемая в соответствии с правилами СУБД (длина, допустимые символы, регистрозависимость).

Сопоставление: Процесс сравнения идентификаторов в зависимости от настроек СУБД (регистрозависимые/независимые, с учётом специальных символов).

Отмена: Удаление/аннулирование идентификатора через оператор DROP или ALTER ... DROP.

Установка: Создание идентификатора при определении объекта (например, CREATE TABLE имя\_таблицы (...)).

Домены (пользовательские типы данных)

Назначение: Определение допустимого набора значений для столбцов, обеспечение семантической целостности данных.

Создание:

```
CREATE DOMAIN имя_домена AS тип_данных [ограничения];
```

Изменение:

```
ALTER DOMAIN имя_домена ...;
```

Удаление:

```
DROP DOMAIN имя_домена [RESTRICT | CASCADE];
```

Особенности ограничений:

- CHECK — задание условий для значений.
- NOT NULL — запрет NULL-значений.
- Ограничения могут быть именованными.

Значения по умолчанию задаются при создании домена:

```
CREATE DOMAIN домен AS тип DEFAULT значение;
```

Правила сравнений символьных типов:

- Зависят от кодировки и параметров БД.
- Регистрозависимость (COLLATE).
- Учёт пробелов (PAD SPACE/NO PAD).
- Использование LIKE, =, <>.

Утверждения (ASSERTIONS)

Назначение: Глобальные ограничения целостности, затрагивающие несколько таблиц или всю БД.

Создание:

```
CREATE ASSERTION имя_утверждения  
CHECK (условие);
```

Удаление:

```
DROP ASSERTION имя_утверждения;
```

Отложенные/не отложенные:

- Не отложенные — проверяются сразу после выполнения оператора.
- Отложенные (DEFERRABLE INITIALLY DEFERRED) — проверяются при фиксации транзакции.

Столбцы типа XML

Oracle:

- Тип данных XMLType.
- Хранится в CLOB (текст) или объектно-реляционно (разобранный).
- Поддержка XQuery, индексов XMLIndex.
- Пример: CREATE TABLE таблица (xml\_col XMLType);

DB2:

- Тип XML.
- Хранится в двоичном формате (внутреннее представление).
- Индексы через XMLPATTERN.
- Пример: CREATE TABLE таблица (xml\_col XML);

SQL Server:

- Тип XML.
- Хранится в внутреннем двоичном формате.
- Индексы: первичный (PRIMARY) и вторичный (PATH, VALUE, PROPERTY).
- Поддержка XSD-схем (коллекции XML-схем).
- Пример: CREATE TABLE таблица (xml\_col XML);

Общие возможности:

- Проверка XSD/XSLT.
- Извлечение данных через XQuery.
- Модификация через updatexml() (Oracle) или modify() (SQL Server).
- Индексирование для ускорения запросов.

**3. Виды ограничений (на атрибуте, кортеже). Где описываются, когда действуют ограничения и когда выполняются. Установка момента проверки ограничений. Уникальные значения: создание, изменение, удаление, начальное значение. Идентифицирующие и вычисляемые столбцы. Примеры реализации идентифицирующих столбцов в Oracle/SQL Server/desktop СУБД.**

Ограничения на атрибуте (столбце):

Применяются к одному столбцу. Определяются при создании/изменении столбца. Примеры: NOT NULL, UNIQUE, PRIMARY KEY, CHECK (на уровне столбца), FOREIGN KEY (ссылка на один столбец).

Ограничения на кортеже (таблице):

Применяются к группе столбцов. Определяются отдельно от описания столбцов. Примеры: PRIMARY KEY (составной), FOREIGN KEY (составной), CHECK (на уровне таблицы, с использованием нескольких столбцов).

Где описываются и когда действуют/выполняются

- В команде CREATE TABLE (в определении столбца или отдельно).
- В команде ALTER TABLE для добавления/изменения ограничений.

Действие: Ограничения действуют постоянно после их создания до удаления или отключения.

Проверка: Ограничения проверяются при выполнении INSERT, UPDATE, DELETE (для FOREIGN KEY также при изменении родительской таблицы).

В SQL существует два основных момента проверки:

- IMMEDIATE: Проверка выполняется сразу после завершения каждой операции изменения данных (по умолчанию в большинстве СУБД).
- DEFERRED (отложенная): Проверка откладывается до конца транзакции.

Устанавливается командой: SET CONSTRAINTS constraint\_name DEFERRED;  
Или при создании ограничения: CONSTRAINT fk\_col FOREIGN KEY (col) REFERENCES tab(id) DEFERRABLE INITIALLY DEFERRED;

Уникальные значения: создание, изменение, удаление, начальное значение  
Создание:

- CREATE TABLE tab (id INT UNIQUE);
- ALTER TABLE tab ADD CONSTRAINT unq\_col UNIQUE (col);

Изменение:

Ограничения UNIQUE нельзя изменить — нужно удалить и создать заново.

Удаление:

ALTER TABLE tab DROP CONSTRAINT unq\_col;

Начальное значение:

Не относится напрямую к ограничению UNIQUE. Для автоматической генерации уникальных значений используются типы данных/объекты: AUTO\_INCREMENT (MySQL), IDENTITY (SQL Server), SEQUENCE (Oracle, PostgreSQL).

Идентифицирующие столбцы (IDENTITY / AUTO\_INCREMENT):

Столбцы с автоматической генерацией уникальных числовых значений при вставке.

Пример (SQL Server):

CREATE TABLE tab (id INT IDENTITY(1,1), name VARCHAR(50));

Вычисляемые столбцы (Computed Columns):

Столбцы, значение которых вычисляется на основе других столбцов таблицы.

Пример (SQL Server):

```
CREATE TABLE tab (  
    price DECIMAL(10,2),  
    tax DECIMAL(10,2),  
    total AS (price + tax) -- вычисляемый столбец  
);
```

Примеры реализации идентифицирующих столбцов в различных СУБД

Oracle:

Используется последовательность (SEQUENCE) и триггеры, либо с версии 12c — IDENTITY.

```
CREATE TABLE tab (  
    id NUMBER GENERATED ALWAYS AS IDENTITY,  
    name VARCHAR2(50)  
);
```

SQL Server:

```
CREATE TABLE tab (  
    id INT IDENTITY(1,1) PRIMARY KEY,  
    data VARCHAR(100)  
);
```

**4. Таблицы: создание, изменение, удаление, конструкции, определяющие столбец, ограничения столбца и таблицы, ссылочная спецификация и её опции и особенности, проверка ограничений, копирование таблиц. Особенности и назначение ограничений Check, References. Реализация рекурсивных связей. Временные таблицы (их виды и размещение) в SQL Server. Вложенные таблицы в Oracle.**

Создание таблиц

```
CREATE TABLE имя_таблицы (  
    столбец1 тип_данных [ограничения],  
    столбец2 тип_данных [ограничения]  
);
```

Изменение таблиц

Добавление столбца:

```
ALTER TABLE Employees  
ADD Email VARCHAR(100);
```

Изменение столбца:

```
ALTER TABLE Employees  
ALTER COLUMN Email VARCHAR(150);
```

Удаление столбца:

```
ALTER TABLE Employees  
DROP COLUMN Email;
```

Добавление ограничения:

```
ALTER TABLE Employees  
ADD CONSTRAINT CHK_BirthDate  
CHECK (BirthDate < GETDATE());
```

Удаление таблиц

```
DROP TABLE Employees;
```

Конструкции, определяющие столбец

- Тип данных: INT, VARCHAR(n), DATE, DECIMAL(p,s).
- Значение по умолчанию: DEFAULT значение
- NULL/NOT NULL: определяет допустимость пустых значений
- IDENTITY: автоматическая генерация значений (SQL Server)
- GENERATED ALWAYS AS: вычисляемые столбцы

Ограничения столбца и таблицы

```
CREATE TABLE Table1 (  
    Col1 INT PRIMARY KEY,  
    Col2 INT NOT NULL,  
    Col3 VARCHAR(50) UNIQUE  
);
```

Ограничения уровня таблицы:

```
CREATE TABLE Table1 (  
    Col1 INT,  
    Col2 INT,  
    Col3 VARCHAR(50),  
    PRIMARY KEY (Col1),  
    UNIQUE (Col3),  
    CONSTRAINT FK_Dept FOREIGN KEY (Col2)  
    REFERENCES Departments(DeptID)  
);
```

Ссылочная спецификация (FOREIGN KEY)

Основной синтаксис:

```
CONSTRAINT имя_ограничения  
FOREIGN KEY (локальный_столбец)  
REFERENCES удаленная_таблица (удаленный_столбец)
```

Опции ON DELETE и ON UPDATE:

- NO ACTION / RESTRICT: Запрет удаления/изменения
- CASCADE: Каскадное удаление/изменение
- SET NULL: Установка NULL в ссылающихся строках
- SET DEFAULT: Установка значения по умолчанию

Проверка ограничений

- Проверка при добавлении/изменении данных
- Отложенная проверка: DEFERRABLE (в некоторых СУБД)

Включение/отключение:

- ALTER TABLE Table1 NOCHECK CONSTRAINT все\_или\_конкретное;
- ALTER TABLE Table1 CHECK CONSTRAINT все\_или\_конкретное;

Копирование таблиц

Структура с данными:

```
SELECT * INTO NewTable FROM OriginalTable;
```

Только структура:

```
SELECT * INTO NewTable FROM OriginalTable WHERE 1=0;
```

Создание на основе запроса:

```
CREATE TABLE NewTable AS  
SELECT Col1, Col2 FROM OriginalTable;
```

Особенности и назначение ограничений

CHECK

Назначение: Проверка значений по условию

Уровни: столбца или таблицы

Пример:

```
CREATE TABLE Products (  
    Price DECIMAL(10,2)  
    CONSTRAINT CHK_Price CHECK (Price > 0),  
    Discount DECIMAL(10,2),  
    CONSTRAINT CHK_Dc CHECK (Discount >= 0 AND Discount <= Price)  
);
```

## REFERENCES (FOREIGN KEY)

Назначение: Обеспечение ссылочной целостности

Особенности:

- Ссылающийся столбец должен соответствовать по типу данных
- Значения должны существовать в родительской таблице
- Может ссылаться на PRIMARY KEY или UNIQUE

Реализация рекурсивных связей

Пример (сотрудник → начальник):

```
CREATE TABLE Employees (  
    EmployeeID INT PRIMARY KEY,  
    FirstName VARCHAR(50),  
    LastName VARCHAR(50),  
    ManagerID INT NULL,  
    CONSTRAINT FK_Manager  
    FOREIGN KEY (ManagerID)  
    REFERENCES Employees(EmployeeID)  
    ON DELETE NO ACTION  
);
```

Локальные временные таблицы:

Префикс: #

Видимость: Только в текущем соединении

Удаление: При закрытии соединения

```
CREATE TABLE #TempTable (ID INT, Name VARCHAR(50));
```

Глобальные временные таблицы:

Префикс: ##

Видимость: Для всех соединений

Удаление: При закрытии всех соединений, ссылающихся на таблицу

```
CREATE TABLE ##GlobalTemp (ID INT, Data VARCHAR(100));
```

Табличные переменные:

Объявление: DECLARE

Видимость: В рамках пакета/процедуры

```
DECLARE @TableVar TABLE (ID INT, Value VARCHAR(50));
```

Таблицы в оперативной памяти (Memory-Optimized):

Хранение: В оперативной памяти

Особенности: Для высокопроизводительных операций

```
CREATE TABLE dbo.FastTable (  
    ID INT NOT NULL PRIMARY KEY NONCLUSTERED,  
    Data VARCHAR(100)  
) WITH (MEMORY_OPTIMIZED = ON);
```

Вложенные таблицы в Oracle

Создание типа коллекции:

```
CREATE TYPE PhoneList AS TABLE OF VARCHAR2(20);
```

Использование в таблице:

```
CREATE TABLE Customers (  
    CustomerID NUMBER PRIMARY KEY,  
    Name VARCHAR2(100),  
    Phones PhoneList  
) NESTED TABLE Phones STORE AS PhoneTable;
```

Вставка:

```
INSERT INTO Customers VALUES (  
    1,  
    'Иванов',  
    PhoneList('123-456', '789-012')  
);
```

Выборка с развертыванием:

```
SELECT c.Name, p.* FROM Customers c, TABLE(c.Phones) p;
```

Обновление:

```
UPDATE TABLE(  
    SELECT Phones FROM Customers WHERE CustomerID = 1  
) p SET VALUE(p) = '999-999' WHERE VALUE(p) = '123-456';
```

Особенности вложенных таблиц:

- Хранение: Отдельно от основной таблицы
- Индексы: Возможно создание индексов на хранимой таблице
- Производительность: Может быть ниже, чем у реляционного дизайна
- Применение: Сложные объектные модели, коллекции данных

**5. Индексы: создание, изменение, удаление, особенности индексов в SQL Server, рекомендации при создании индексов: правила выбора таблиц и столбцов, составных индексов. Представления: создание, удаление, критерии обновляемости, особенности в SQL Server. Схемы: создание, удаление, установка.**

Индекс — это структура данных в бд, которая ускоряет операции поиска и сортировки за счет дополнительных затрат на дисковое пространство и обслуживание.

Создание индекса

- Кластерный индекс (только один на таблицу)  
CREATE CLUSTERED INDEX IX\_Таблица\_Поле ON Таблица (Поле);
- Некластерный индекс  
CREATE [UNIQUE] [NONCLUSTERED] INDEX IX\_Таблица\_Поля  
ON Таблица (Поле1 [ASC|DESC], Поле2)  
INCLUDE (ДопПоле1, ДопПоле2) -- Включенные столбцы  
WHERE Условие; -- Фильтрованный индекс (SQL Server)
- Индекс по представлению (требует SCHEMABINDING)  
CREATE UNIQUE CLUSTERED INDEX IX\_Представление  
ON Представление(Поле);

## Изменение индекса

Прямого изменения структуры индекса нет. Необходимо удалить и создать заново или использовать ALTER INDEX для обслуживания:

Перестроение индекса: ALTER INDEX IX\_Имя ON Таблица REBUILD;

Реорганизация индекса: ALTER INDEX IX\_Имя ON Таблица REORGANIZE;

Отключение индекса: ALTER INDEX IX\_Имя ON Таблица DISABLE;

Удаление индекса: DROP INDEX IX\_Имя ON Таблица;

## Особенности индексов в SQL Server

- Кластерный индекс определяет физический порядок данных
- Columnstore-индексы для аналитических запросов
- Фильтрованные индексы для части данных (WHERE условие)
- Индексы с включенными столбцами для покрывающих индексов
- Географические и XML-индексы для специализированных типов данных
- Параметры хранения: FILLFACTOR, PAD\_INDEX, SORT\_IN\_TEMPDB
- Онлайн-построение индексов (Enterprise Edition) без блокировок

## Рекомендации при создании индексов

Выбор таблиц:

- Таблицы с большим объемом данных
- Часто используемые в запросах JOIN/WHERE
- Таблицы с частыми операциями обновления требуют осторожности

Выбор столбцов:

- Высокая селективность (уникальность >90%)
- Часто используемые в условиях WHERE, JOIN, ORDER BY, GROUP BY
- Внешние ключи
- Избегать индексирования:
  - Часто изменяемых столбцов
  - Столбцов с низкой селективностью (пол, статус)
  - Слишком многих столбцов в одном индексе (оптимально 1-3)

## Составные индексы:

- Порядок столбцов критически важен (от наиболее селективного к наименее)
- Правило левого префикса: (A,B,C) работает для запросов с (A),(A,B),(A,B,C)
- Включенные столбцы (INCLUDE) для покрывающих запросов
- Учитывать порядок сортировки (ASC/DESC) для ORDER BY

## Представления (Views) в SQL Server

Создание представления:

```
CREATE [OR ALTER] VIEW [Схема].Имя_Представления  
[WITH SCHEMABINDING|ENCRYPTION|VIEW_METADATA]  
AS
```

```
    SELECT Запрос
```

```
    [WITH CHECK OPTION]; -- Запрет модификации строк, не видимых
```

через представление

Удаление представления:

```
DROP VIEW [Схема].Имя_Представления;
```

Представление обновляемо, если:

- Запрос содержит только одну таблицу в FROM (без JOIN)
- Нет агрегатных функций (COUNT, SUM, AVG и т.д.)
- Нет DISTINCT, GROUP BY, HAVING
- Нет оконных функций
- Нет UNION, EXCEPT, INTERSECT
- Не содержит вложенных запросов в SELECT
- Все не-NULL столбцы базовой таблицы включены в представление (для INSERT)

Искл: INSTEAD OF триггеры позволяют обновлять любые представления

Особенности в SQL Server

- Индексируемые представления (требуют SCHEMABINDING, уникальный кластерный индекс)
- Секционированные представления (распределение данных по таблицам)
- Системные представления (каталог): sys.objects, sys.tables
- Динамическое управление представлениями (DMV) для мониторинга
- WITH CHECK OPTION для контроля целостности
- ENCRYPTION для шифрования определения представления

Схема — это (пространство имён) в базе данных, который группирует связанные объекты (таблицы, представления, процедуры) для организации, управления доступом и устранения конфликтов имён.

Создание схемы: CREATE SCHEMA [Имя] [AUTHORIZATION Владелец];

Удаление схемы: DROP SCHEMA [Имя\_Схемы]; Удаляет только пустую схему

Установка/Использование схем

Изменение схемы объекта ALTER SCHEMA Нов\_Сх TRANSFER Стар\_Сх.Обкт;

Указание схемы в запросах SELECT \* FROM Схема.Таблица;

Изменение схемы по умолчанию для пользователя:

```
ALTER USER Пользователь  
WITH DEFAULT_SCHEMA = Схема;
```

**6. Роли: создание, назначение. Предоставление и отмена прав доступа к объектам. Системные и объектные привилегии. Установка атрибутов (уровней изоляции) транзакции, проблемы параллелизма, фиксация и отмена изменений. Журналы транзакций. Точки сохранения. Особенности транзакций, включая распределённые, в SQL Server. Сеансы: установление, разрыв и переключение связи. Управление транзакциями и сессиями компонентами Delphi. Особенности предоставления и отмены прав в SQL Server.**

Создание роли в SQL: CREATE ROLE <имя\_роли>;

Назначение роли: GRANT <имя\_роли> TO <пользователь>;

Объектные привилегии: Действуют в отношении конкретных объектов бд. Основные: SELECT, INSERT, UPDATE, DELETE, EXECUTE, REFERENCES.

Предоставление: GRANT <привилегия> ON <объект> TO <пользователь/роль>;

Отмена: REVOKE <привилегия> ON <объект> FROM <пользователь/роль>;.

Системные привилегии: Относятся к операциям на уровне бд или экземпляра (например, CREATE TABLE, BACKUP DATABASE).

Предоставление: GRANT <привилегия> TO <пользователь/роль>;

Отмена: REVOKE <привилегия> FROM <пользователь/роль>;

Уровни изоляции (SET TRANSACTION ISOLATION LEVEL):

- READ UNCOMMITTED (чтение незафиксированных данных).
- READ COMMITTED (чтение зафиксированных данных) – по умолчанию.
- REPEATABLE READ (повторяемое чтение).
- SERIALIZABLE (упорядочиваемость).
- SNAPSHOT (моментальный снимок) – в SQL Server.

Проблемы параллелизма: "Грязное" чтение, неповторяемое чтение, фантомное чтение, потерянное обновление.

Фиксация и отмена: COMMIT – подтверждение транзакции. ROLLBACK – откат всех изменений транзакции к началу или к точке сохранения.

Журнал транзакций (Transaction Log): Служебный файл, в который записываются все изменения данных и служебная информация перед их физической записью в базу. Обеспечивает атомарность, согласованность, возможность восстановления.

Точка сохранения (SAVEPOINT): Позволяет откатить часть транзакции.

- Создание: SAVE TRANSACTION <имя\_точки>;
- Откат к точке: ROLLBACK TRANSACTION <имя\_точки>;

Особенности транзакций, включая распределённые, в SQL Server.

- Локальные транзакции: Общаются одним экземпляром SQL Server.
- Распределённые транзакции: Затрагивают два и более сервера. Координация выполняется с помощью координатора распределённых транзакций. Используются инструкции BEGIN DISTRIBUTED TRANSACTION или неявно, когда транзакция ссылается на связанные серверы.
- Автоматически управляемые (неявные) и явные транзакции: По умолчанию включен режим AUTOCOMMIT. Для явного управления: BEGIN TRANSACTION, COMMIT, ROLLBACK.
- Вложенные транзакции: SQL Server поддерживает вложенность. Фактический COMMIT происходит только на верхнем уровне. ROLLBACK на верхнем уровне откатывает всю транзакцию.
- Установление сеанса: Физическое подключение клиентского приложения к серверу БД. В SQL Server сеансу присваивается уникальный SPID.
- Разрыв сеанса: Закрытие приложения или выполнение команды KILL <SPID> для принудительного завершения.
- Переключение связи: Техника, при которой физическое соединение с БД после закрытия логического сеанса не уничтожается, а возвращается в пул для повторного использования новым логическим сеансом.

Управление транзакциями и сессиями компонентами Delphi.

Компоненты: Используются, как правило, компоненты доступа к данным (dbGo, FireDAC, UniDAC).

Сессии:

- Компонент TSession или его аналоги управляют группой подключений.
- В FireDAC: TFDConnection представляет собой сеанс/подключение. Управление: Open(), Close(), параметры ConnectionString.

Транзакции: Управление осуществляется через методы объекта подключения:

- Начало: StartTransaction.
- Фиксация: Commit.
- Откат: Rollback.

Особенности предоставления и отмена прав в SQL Server.

- Иерархия субъектов безопасности: Серверные имена входа (LOGIN) -> Пользователи бд (USER) -> Роли базы данных (ROLE) -> Члены ролей.
- Схемы (SCHEMA): Привилегии часто назначаются на уровне схемы, а не отдельных таблиц. GRANT SELECT ON SCHEMA::Sales TO accountant;
- Исполнение (EXECUTE): Особое внимание правам на выполнение хранимых процедур и функций.
- Запрет (DENY): SQL Server имеет расширенную команду DENY, которая явно запрещает привилегию и имеет приоритет над GRANT. Это мощный механизм для закрытия доступа.
- Владелец (Ownership): Владелец объекта (например, схемы) имеет на него все права. Цепочки владения могут упрощать управление правами.
- Роли сервера: Существуют предопределённые серверные роли (например, sysadmin, serveradmin), управляемые через GRANT/DENY.

**7. Конструкторы значений строк и таблиц. Добавление и удаление строк таблицы. Загрузка LOB в Oracle/DB2/SQL Server. Изменение данных в таблицах. Использование данных из курсоров. Слияние данных. Использование Null-значений. Особенности операторов изменения данных в Oracle/SQL Server. Ограничение числа обрабатываемых строк.**

- Конструктор строк: Используется для создания строки данных. В SQL: ROW(1, 'text') или VALUES (1, 'text').
- Конструктор таблиц: Позволяет создавать временные таблицы в запросе. SQL Server: SELECT \* FROM (VALUES (1, 'A'), (2, 'B')) AS t(id, name);

Добавление строк таблицы: INSERT INTO t\_name (col1, col2) VALUES (v1, v2);

Удаление: DELETE FROM table\_name WHERE condition;

Загрузка LOB:

- Oracle: Используется BFILE, BLOB, CLOB. Загрузка через DBMS\_LOB или INSERT с указанием данных.
- DB2: Поддержка BLOB, CLOB, DBCLOB. Загрузка через LOAD/INSERT.
- SQL Server: Типы VARBINARY(MAX), VARCHAR(MAX), NVARCHAR(MAX). Загрузка через OPENROWSET или INSERT.

Изменение данных в таблице: UPDATE t\_name SET col1 = val1 WHERE cond;

Курсоры позволяют обрабатывать строки по одной. Например, в PL/SQL:

```
DECLARE CURSOR cur IS SELECT * FROM table;
BEGIN
  FOR row IN cur LOOP
    -- обработка row
  END LOOP;
END;
```

Слияние данных

Оператор MERGE (UPSERT): Объединяет INSERT, UPDATE, DELETE в одной операции. Например:

```
MERGE INTO target USING source ON condition
WHEN MATCHED THEN UPDATE SET ...
WHEN NOT MATCHED THEN INSERT ...;
```

NULL представляет отсутствие данных. Проверка: IS NULL или IS NOT NULL. В операциях NULL приводит к NULL (5 + NULL = NULL).

Особенности операторов изменения данных в Oracle/SQL Server

Oracle:

- RETURNING INTO для возврата значений после INSERT/UPDATE/DELETE.
- MERGE с поддержкой DELETE.

SQL Server:

- OUTPUT для возврата измененных данных.
- MERGE с ограничениями на триггеры.

Ограничение числа обрабатываемых строк

SQL Server: TOP n/FETCH FIRST n ROWS: DELETE TOP (10) FROM t\_name;

Oracle: ROWNUM/FETCH FIRST: DELETE FROM t\_n WHERE ROWNUM < 10;

DB2: FETCH FIRST n ROWS ONLY.

**8. Раздел Select оператора Select: правила включения строк, ключевые слова, включение констант, агрегатные функции и особенности их использования, именование выходных колонок, вычисляемые колонки, подзапросы в разделе Select, способы именования и подсказки в СУБД Oracle.**

Строки включаются на основе условий WHERE, GROUP BY, HAVING и JOIN. DISTINCT исключает дубликаты, ALL (по умолч) сохраняет все дубликаты.

Ключевые слова

- SELECT – перечень столбцов или выражений.
- DISTINCT / UNIQUE – уникальные строки.
- \* – выбор всех столбцов таблицы.
- AS – псевдоним для столбцов или таблиц.
- FROM – указание источника данных.

Включение констант

Константы включаются напрямую в список выборки:

```
SELECT 'Константа', column1 FROM table;
```

Агрегатные функции: SUM(), AVG(), COUNT(), MAX(), MIN(), STDDEV(), VARIANCE().

- Применяются к группам строк, определенным GROUP BY.
- COUNT(\*) считает все строки; COUNT(column) игнорирует NULL.
- Без GROUP BY возвращают одну строку для всего набора.
- В разделе SELECT с агрегатными функциями неагрегированные столбцы должны быть в GROUP BY.
- HAVING фильтрует результаты агрегации (аналог WHERE для групп).

Именованые выходные колонки происходят через AS или пробел:

```
SELECT column1 AS name1, column2 name2 FROM table;
```

Вычисляемые колонки: выражения на основе столбцов, констант и функций:

```
SELECT price * quantity AS total, UPPER(name) FROM table;
```

Подзапросы в разделе SELECT

- Скалярные подзапросы возвращают одно значение
- SELECT name, (SELECT MAX(price) FROM products) AS mx\_pr FROM users;
- Должны возвращать одну строку и один столбец.
- Подзапрос возвращает >1 строки – ошибка (кроме ANY, ALL в сравнении).
- Могут быть коррелированными (связаны с внешним запросом).

Способы именования и подсказки в СУБД Oracle

Именованые:

- Псевдонимы таблиц: FROM table t.
- Кавычки для регистрозависимых имен: "ColumnName".
- Псевдонимы столбцов: SELECT column AS alias.

Подсказки (hints) – указания оптимизатору в /\*+ ... \*/:

- /\*+ INDEX(table index) \*/ – использование индекса.
- /\*+ FULL(table) \*/ – полное сканирование таблицы.
- /\*+ ORDERED \*/ – порядок соединения таблиц.
- /\*+ PARALLEL(table, n) \*/ – параллельное выполнение.

**9. Раздел From оператора Select: правила указания таблиц, варианты соединений в разделе From. Картезианские соединения, самообъединения. Соединения Cross|Outer Apply. Подзапросы в разделе From. Встроенные функции SQL.**

В разделе FROM указываются источники данных для запроса.

Основные объекты:

- Имя таблицы/представления: FROM ObjName
- Псевдонимы: Для удобства таблицам можно присваивать псевдонимы с помощью AS или пробела: FROM TableName AS T или FROM TableName T.

## Варианты соединений в разделе FROM

Соединения определяют, как строки из 2+ таблиц связываются между собой.

- INNER JOIN: Возвращает строки, для которых есть соответствие.
- LEFT JOIN: Возвращает все строки из левой таблицы и соотв строки из правой. При отсутствии соответствия в правой возвращаются NULL.
- RIGHT JOIN: Возвращает все строки из правой таблицы и соответствующие строки из левой.
- FULL JOIN: Возвращает все строки, когда есть совпадение либо в левой, либо в правой таблице. Недостающие столбцы заполняются NULL.
- CROSS JOIN: См. ниже.

Картезианские соединения: Возвращает декартово произведение двух таблиц. Каждая строка первой таблицы соединяется с каждой строкой второй таблицы.

Синтаксис:

- SELECT \* FROM Table1 CROSS JOIN Table2;
- SELECT \* FROM Table1, Table2;

Самообъединения: Соединение таблицы с самой собой. Требуется для сравнения строк внутри одной таблицы.

Обязательное условие: Использование псевдонимов для логического разделения таблицы на две "разные".

Найти сотрудников и их непосредственных руководителей:

```
SELECT E.EmployeeName, M.EmployeeName AS ManagerName
FROM Employees E
LEFT JOIN Employees M ON E.ManagerID = M.EmployeeID;
```

## Соединения CROSS APPLY и OUTER APPLY

- Особенность: Позволяют вызывать табличную функцию или подзапрос для каждой строки левой таблицы. Правая часть зависит от левой.
- CROSS APPLY: Работает аналогично INNER JOIN. Возвращает строки из левой таблицы, для которых вызов правой части вернул результирующий набор.  
SELECT \* FROM Departments D  
CROSS APPLY dbo.GetEmployeesByDepartment(D.Id) E;
- OUTER APPLY: Работает аналогично LEFT JOIN. Возвращает все строки из левой таблицы, даже если правая часть вернула пустой набор (столбцы из правой части будут NULL).
- Отличие от JOIN: В JOIN правая часть вычисляется один раз. В APPLY правая часть вычисляется для каждой строки левой таблицы.

Подзапросы в разделе FROM: Результат подзапроса (SELECT) используется в разделе FROM как временная таблица.

Обязательное условие: Должен иметь псевдоним.

```
SELECT A.DepartmentId, AVG(A.Salary) AS AvgSalary
FROM (
    SELECT DepartmentId, Salary FROM Employees WHERE Salary > 30000
) AS A
GROUP BY A.DepartmentId;
```

Встроенные функции SQL в разделе FROM: Функции, возвращающие табличный тип данных, могут быть указаны непосредственно в FROM.

Типы функций:

- Встроенные табличные функции: Содержат один оператор RETURN SELECT.... Обработываются оптимизатором как представление.  
CREATE FUNCTION GetEmployees(@DeptId INT)  
RETURNS TABLE  
AS  
RETURN (SELECT \* FROM Employees WHERE DepartmentId = @DeptId);
- Многооператорные табличные функции: Содержат логику в теле функции, создают и заполняют возвращаемую таблицу явно.
- Связь с APPLY: Часто используются вместе с операторами CROSS APPLY и OUTER APPLY, когда функция зависит от столбцов внешней таблицы.

**10. Раздел Where оператора Select: установка связи между таблицами, эквисоединения, тэта-соединения, предикаты: сравнения, попадания во множество, принадлежности диапазону, подобия, проверки отсутствующих значений, уникальности, совпадения, перекрытия, булевы операторы. Особенности предикатов и соединений в Oracle/SQL Server.**

Связь между таблицами в WHERE реализуется через указание условия соединения. Ранее это был основной способ, но сейчас чаще используется синтаксис JOIN ... ON.

Эквисоединение: Соединение по равенству ключей.

SELECT \* FROM Employees e, Departments d WHERE e.dept\_id = d.dept\_id;

Тета-соединение (Theta-join): Соединение по условному отношению (=, >, <...).

SELECT \* FROM Salaries s1, Salaries s2 WHERE s1.amount > s2.amount;

Предикаты в разделе WHERE

- Предикат сравнения: =, >, <, >=, <=, <> (или !=).
- Oracle: Для сравнения с NULL используется IS NULL или IS NOT NULL.
- SQL Server: Аналогично, = NULL не работает, требуется IS NULL.
- Предикат попадания во множество (IN): WHERE status IN ('Active', 'Pending')
- Предикат принадлежности диапазону (BETWEEN) (Включает границы):  
WHERE salary BETWEEN 30000 AND 50000;
- Предикат подобия (LIKE):
  - % - любое количество любых символов.
  - \_ - ровно один любой символ.
- Предикат проверки отсутствующих значений (IS [NOT] NULL):
- Предикат совпадения (EXISTS): Проверяет, возвращает ли подзапрос что-то.  
WHERE EXISTS (SELECT 1 FROM Orders o WHERE o.cust\_id = c.id)
- Предикат перекрытия (OVERLAPS): Для проверки пересечения интервалов дат/времени.  
WHERE (start1, end1) OVERLAPS (start2, end2)
- Булевы операторы: AND, OR, NOT.

## Особенности в Oracle и SQL Server

### Oracle:

- Предикат DECODE в WHERE: Функцию DECODE можно использовать для простых сравнений по принципу "ключ-значение".
- Обработка NULL в сравнениях: Любое сравнение с NULL (кроме IS NULL) дает UNKNOWN, что в WHERE трактуется как FALSE.

### SQL Server:

- Предикат LIKE с наборами и диапазонами: Поддерживает [A-C] (один символ от A до C) и [^A-C] (не A-C).
- Различия в сравнении строк: Поведение зависит от параметров сортировки. Может быть чувствительным/нечувствительным к регистру, акцентам.
- Оператор != vs <>: Оба работают, <> является стандартом.
- Функция IIF в WHERE: Может использоваться для простых условий прямо в предикате (с SQL Server 2012).

WHERE IIF(status = 'A', active\_date, archive\_date) > @cutoff

## **11. Раздел Where оператора Select: вложение запросов, коррелирующие запросы, условия поиска во вложенном запросе, ограничения на запросы в подзапросах, предикаты существования, количественного сравнения. Рекурсивные запросы и их назначение. Особенности соединения таблиц в SQL Server.**

### Вложение запросов (Подзапросы):

Подзапрос – это запрос, вложенный в другой (чаще в WHERE, FROM, SELECT). Выполняется первым, и его результат используется внешним запросом.

```
SELECT * FROM Employees
WHERE DepartmentID =
    (SELECT DepartmentID FROM Departments WHERE Name = 'IT');
```

Коррелирующие запросы: подзапрос, который ссылается на столбцы внешнего запроса. Выполняется для каждой строки внешнего запроса.

```
SELECT * FROM Employees e1
WHERE Salary >
    (SELECT AVG(Salary) FROM Employees e2
    WHERE e2.DepartmentID = e1.DepartmentID);
```

Условия поиска в подзапросе: результат подзапроса можно использовать в условиях внешнего запроса с операторами =, >, IN, ANY/ALL, EXISTS.

### Ограничения на запросы в подзапросах:

- Количество и тип данных столбцов должны соответствовать условию внешнего запроса.
- Подзапрос в SELECT (как столбец) должен возвращать скалярное значение.

Предикат существования EXISTS: Проверяет существование строк, возвращаемых подзапросом. Возвращает TRUE, если подзапрос вернул хотя бы одну строку. Пример (отделы, в которых есть сотрудники):

Предикаты количественного сравнения (ANY/SOME и ALL): Сравнивают значение с каждым значением, возвращаемым подзапросом.

- ANY/SOME: Условие истинно, если оно верно хотя бы для одного значения из набора подзапроса.
- ALL: Условие истинно, если оно верно для всех значений из набора подзапроса.

```
SELECT * FROM Employees
```

```
WHERE Salary > ALL
```

```
(SELECT Salary FROM Employees WHERE Position = 'Manager');
```

Рекурсивный запрос – это запрос, который вызывает сам себя. Реализуется с помощью обобщенных табличных выражений.

Назначение: Обработка иерархических или древовидных структур данных.

Структура: состоит из двух частей, объединенных UNION ALL:

Якорная часть: Определяет начальную точку рекурсии (корень иерархии).

Рекурсивная часть: Выполняется итеративно, используя предыдущий результат.

WITH EmployeeHierarchy AS (

-- Анкор: начальник (у которого нет менеджера)

```
SELECT EmployeeID, FullName, ManagerID, 1 AS Level
```

```
FROM Employees WHERE ManagerID IS NULL
```

```
UNION ALL
```

-- Рекурсия: подчиненные

```
SELECT e.EmployeeID, e.FullName, e.ManagerID, eh.Level + 1
```

```
FROM Employees e
```

```
INNER JOIN EmployeeHierarchy eh ON e.ManagerID = eh.EmployeeID
```

```
)
```

```
SELECT * FROM EmployeeHierarchy;
```

Особенности соединения таблиц в SQL Server

Поддерживаемые типы JOIN:

- INNER JOIN (внутреннее) – по умолчанию.
- LEFT [OUTER] JOIN (левое внешнее).
- RIGHT [OUTER] JOIN (правое внешнее).
- FULL [OUTER] JOIN (полное внешнее).
- CROSS JOIN (перекрестное).

Синтаксис: Поддерживается как старый стандарт FROM T1, T2 WHERE ..., так и явный современный синтаксис FROM T1 JOIN T2 ON ....

Особенности при соединении:

- Можно соединять более двух таблиц в одном запросе.
- Возможно соединение таблицы с самой собой для иерархических сравнений.
- CROSS APPLY и OUTER APPLY: Специфичные операторы, позволяющие вызывать табличную функцию для каждой строки левой таблицы.

**12. Раздел Group By оператора Select: разновидности группирования, их особенности, раздел Having и виды его предикатов. Принцип определения выходных колонок при группировании. Особенности группирования запросов в Oracle/SQL Server. Раздел Order By оператора Select и его особенности в Oracle.**

GROUP BY используется для группировки строк по значениям указанных колонок с последующим применением агрегатных функций.

Разновидности группирования:

- Простая группировка: группировка по одному или нескольким столбцам.
- Группировка с выражениями  

```
SELECT EXTRACT(YEAR FROM hire_date), COUNT(*)  
FROM employees GROUP BY EXTRACT(YEAR FROM hire_date);
```
- Расширенная группировка (расширения ROLLUP, CUBE, GROUPING SETS)
  - ROLLUP: Создает иерархические итоги (подытоги и общий итог).
  - CUBE: Генерирует все возможные комбинации группировок.
  - GROUPING SETS: Позволяет задать конкретные наборы группировок.

Раздел HAVING и виды его предикатов

- HAVING применяется для фильтрации результатов группировки.
- Используется после GROUP BY.
- Может содержать агрегатные функции.

Виды предикатов в HAVING:

- Простые условия с агрегатами
- Комбинированные условия с агрегатами
- Предикаты с вложенными подзапросами

Принцип определения выходных колонок при группировании

- В SELECT при использовании GROUP BY можно указывать:
- Колонки, перечисленные в GROUP BY.
- Агрегатные функции (COUNT, SUM, AVG и т.д.).
- Нельзя включать неагрегированные колонки, не входящие в GROUP BY.

Особенности группирования запросов в Oracle/SQL Server

Oracle:

- Поддержка расширенных группировок (ROLLUP, CUBE, GROUPING SETS).
- Функция GROUPING() для идентификации строк итогов в результатах ROLLUP/CUBE.
- Возможность использования аналитических функций (например, OVER()) без группировки.

SQL Server:

- Аналогичная поддержка ROLLUP, CUBE, GROUPING SETS (с синтаксисом, совместимым со стандартом SQL).
- Функция GROUPING\_ID() для определения уровня агрегации.
- Индексированные представления для оптимизации группировок.

ORDER BY и его особенности в Oracle

- Можно указывать порядковый номер колонки из SELECT (начиная с 1).  
SELECT first\_name, last\_name  
FROM employees  
ORDER BY 1; -- Сортировка по first\_name
- Сортировка NULL-значений: По умолчанию NULL считаются наибольшими значениями. Можно управлять через NULLS FIRST или NULLS LAST:  
SELECT last\_name, commission\_pct  
FROM employees  
ORDER BY commission\_pct NULLS FIRST;
- Сортировка по выражению или функции

**13. Разделы Union, Intersect, Except оператора Select: ограничения, ключевые слова, различия, сортировка. Организация ветвлений, работа с Null-значениями и мультимножествами. Особенности Null-значений в предикатах. Оконные функции. Ограничения числа возвращаемых строк в запросе. Общие правила написания эффективных запросов.**

- UNION — объединение результатов двух или более запросов. Удаляет дубликаты строк, если не используется UNION ALL.
- INTERSECT — пересечение результатов. Возвращает только строки, которые присутствуют в результатах всех запросов. Удаляет дубликаты.
- EXCEPT — разность результатов. Возвращает строки из первого запроса, которых нет в результатах второго запроса. Удаляет дубликаты.

Основное ограничение: Количество и порядок столбцов во всех объединяемых запросах должны быть одинаковыми. Типы данных соответствующих столбцов должны быть совместимыми.

Сортировка: ORDER BY можно применить только к результату операции. Указывать его можно только в конце последнего запроса. Сортировка по порядковому номеру столбца ссылается на столбцы результирующего набора.

Ветвления реализуются на уровне выражений с помощью оператора CASE (простой CASE WHEN ... THEN ... или поисковый CASE WHEN условие THEN ...). На уровне запросов — через UNION/UNION ALL разных наборов строк в зависимости от условий в WHERE.

NULL — это специальный маркер, обозначающий отсутствие, неизвестность или неприменимость значения. Любая операция сравнения или арифметическая операция с NULL возвращает NULL. Для проверки на NULL всегда используется оператор IS NULL или IS NOT NULL.

Мультимножества: Результат запроса без DISTINCT по умолчанию является мультимножеством (набор строк, где дубликаты разрешены).

UNION, INTERSECT, EXCEPT по умолчанию работают как операции над множествами (удаляют дубликаты). UNION ALL, INTERSECT ALL, EXCEPT ALL работают как операции над мультимножествами.

Особенности NULL-значений в предикатах: Условие с NULL оценивается не как TRUE/FALSE, а как UNKNOWN.

Фильтрация в WHERE: Строка включается в результат только если условие в WHERE равно TRUE. Значения FALSE и UNKNOWN отбрасываются.

Операторы IN и NOT IN с NULL: value IN (1, 2, NULL) эквивалентно value = 1 OR value = 2 OR value = NULL. Если value не равен 1 или 2, результат будет UNKNOWN, и строка не войдет в выборку.

value NOT IN (1, 2, NULL) эквивалентно value <> 1 AND value <> 2 AND value <> NULL. Если в списке есть NULL, часть value <> NULL всегда дает UNKNOWN, а AND с UNKNOWN дает UNKNOWN. Следовательно, весь предикат никогда не будет TRUE, и запрос может вернуть пустой результат, даже если ожидалась строка.

Оконные функции: вычисление агрегатных, ранжирующих и статистических значений над окном (группой строк), связанных с текущей строкой, без свертывания строк в одну, в отличие от GROUP BY.

Синтаксис: Функция(...) OVER (PARTITION BY ... ORDER BY ... [frame\_clause])

- PARTITION BY — определяет группы строк, внутри которых происходит вычисление.
- ORDER BY — задает порядок строк внутри раздела.
- frame\_clause (ROWS/RANGE BETWEEN ... AND ...) — точно определяет набор строк внутри раздела для каждой текущей строки (например, "3 строки перед текущей и 1 после").

Примеры функций: ROW\_NUMBER(), RANK(), SUM(...), AVG(...), LAG(...), LEAD(...).

Ограничение числа возвращаемых строк в запросе

Стандартный SQL: FETCH FIRST n ROWS ONLY

T-SQL (MS SQL Server): TOP (n) [PERCENT] в начале SELECT.

Общие правила написания эффективных запросов:

- Выбирайте только нужные столбцы: Избегайте SELECT \*. Явно перечисляйте необходимые столбцы.
- Ограничивайте объем данных на раннем этапе: Используйте WHERE как можно раньше, особенно перед JOIN и группировки GROUP BY.
- Используйте EXISTS вместо IN для подзапросов.
- Корректно применяйте JOIN: Выбирайте минимально необходимый тип JOIN. Следите за порядком таблиц в JOIN и условиями соединения. Используйте индексированные столбцы для соединения.
- Будьте осторожны с функциями в условиях WHERE: Использование функций над столбцом (WHERE UPPER(name) = 'ИВАН') часто предотвращает использование индексов.
- Минимизируйте количество обращений к БД: Объединяйте запросы, где это возможно, используйте временные таблицы/обобщенные табличные выражения (CTE) для многоэтапной обработки сложных данных.

**14. SQL/PSM. Многооператорные процедуры и атомарность, объявление переменных и присвоение им значений, управление логикой исполнения, операторы циклов. Указатели условий (создание и обработка). Обработка ошибок в Oracle/SQL Server. Оператор неявного открытия курсора. Особенности работы с переменными, циклами, ветвлениями в Oracle.**

В SQL/PSM процедуры могут содержать несколько операторов, объединенных в логический блок. Атомарность обеспечивается транзакциями:

- Oracle: По умолчанию каждая операция автофиксируется. Для атомарности используют BEGIN ... END с управлением транзакциями через COMMIT/ROLLBACK.
- SQL Server: Процедуры выполняются в транзакции. Атомарность контролируется через BEGIN TRANSACTION, COMMIT, ROLLBACK.

Объявление переменных и присвоение им значений

Oracle PL/SQL:

```
DECLARE
    var_name NUMBER(10) := 100; -- Инициализация
    var_text VARCHAR2(50);
BEGIN
    var_text := 'Пример';
    SELECT column INTO var_name FROM table WHERE ...;
END;
```

SQL Server T-SQL:

```
DECLARE @var_name INT = 100;
DECLARE @var_text VARCHAR(50);
SET @var_text = 'Пример';
SELECT @var_name = column FROM table WHERE ...;
```

Ветвления:

- Oracle: IF ... THEN ... ELSIF ... ELSE ... END IF;
- SQL Server: IF ... BEGIN ... END ELSE BEGIN ... END.

Циклы:

Oracle:

```
LOOP
    EXIT WHEN условие;
END LOOP;

WHILE условие LOOP ... END LOOP;

FOR i IN 1..10 LOOP ... END LOOP;
```

SQL Server:

```
WHILE условие
BEGIN ... END;
```

Указатели условий

Oracle: Курсоры для обработки наборов данных:

```
DECLARE
  CURSOR cur_name IS SELECT ... FROM ...;
  rec cur_name%ROWTYPE;
BEGIN
  OPEN cur_name;
  LOOP
    FETCH cur_name INTO rec;
    EXIT WHEN cur_name%NOTFOUND;
    -- Обработка записи
  END LOOP;
  CLOSE cur_name;
END;
```

SQL Server:

```
DECLARE cur_name CURSOR FOR SELECT ... FROM ...;
OPEN cur_name;
FETCH NEXT FROM cur_name INTO @var1, @var2;
WHILE @@FETCH_STATUS = 0
BEGIN
  -- Обработка
  FETCH NEXT FROM cur_name INTO @var1, @var2;
END;
CLOSE cur_name;
DEALLOCATE cur_name;
```

Обработка ошибок

Oracle PL/SQL:

```
BEGIN
  -- Операции
EXCEPTION
  WHEN NO_DATA_FOUND THEN ...;
  WHEN OTHERS THEN
    RAISE_APPLICATION_ERROR(-20001, 'Ошибка: ' || SQLERRM);
END;
```

SQL Server T-SQL:

```
BEGIN TRY
  -- Операции
END TRY
BEGIN CATCH
  PRINT 'Ошибка: ' + ERROR_MESSAGE();
  THROW;
END CATCH;
```

Оператор неявного открытия курсора

Oracle: Неявный курсор используется в FOR циклах:

SQL Server: Курсор в FETCH цикле, но неявные курсоры менее распространены.

Особенности работы с переменными, циклами, ветвлениями в Oracle

Переменные:

- Строгая типизация, возможно использование %TYPE и %ROWTYPE.
- Инициализация при объявлении.

Циклы:

- Цикл FOR автоматически открывает/закрывает курсор.
- CONTINUE и EXIT для управления итерациями.

Ветвления:

- Условия в IF должны возвращать TRUE/FALSE/NULL.
- Поддержка CASE в SQL-запросах и PL/SQL.

**15. SQL/PSM. Хранимые процедуры: создание, удаление, изменение, объявление параметров, вызов, полиморфизм, предоставление привилегий на исполнение. Типы данных. Достоинства и недостатки использования процедур. Процедуры PL/SQL. Особенности работы с хранимыми процедурами в Oracle/SQL Server. Системные процедуры.**

Процедура создается оператором CREATE PROCEDURE.

SQL/PSM, DB2, MySQL

```
CREATE PROCEDURE имя ([IN|OUT|INOUT] параметр тип, ...)
BEGIN
    -- тело процедуры
END;
```

PL/SQL (Oracle)

```
CREATE [OR REPLACE] PROCEDURE имя_процедуры (
    параметр1 IN тип,
    параметр2 OUT тип,
    ...
) AS
BEGIN
    -- тело процедуры
END;
```

T-SQL (SQL Server)

```
CREATE PROCEDURE имя_процедуры
    @параметр1 тип,
    @параметр2 тип OUTPUT,
    ...
AS
BEGIN
    -- тело процедуры
END;
```

Удаление процедуры: DROP PROCEDURE имя\_процедуры; -- везде

Изменение процедуры:

Oracle: REPLACE PROCEDURE

SQL Server: ALTER PROCEDURE

MySQL: ALTER PROCEDURE

## Объявление параметров

Параметры могут быть трех видов:

- IN (входной, по умолчанию) — для передачи значения в процедуру.
- OUT (выходной) — для возврата значения из процедуры.
- INOUT — комбинированный.

Oracle PL/SQL:

```
CREATE PROCEDURE add_employee (  
    p_id IN NUMBER,  
    p_name IN VARCHAR2,  
    p_salary OUT NUMBER  
) AS ...
```

## Вызов процедуры

```
BEGIN -- В анонимном блоке (Oracle PL/SQL)  
    имя_процедуры(параметр1, параметр2);  
END;
```

```
EXEC имя @пар1, @пар2 OUTPUT; -- В T-SQL (SQL Server)
```

```
CALL имя_процедуры(пар1, пар2); -- В стандарте SQL (CALL)
```

## Полиморфизм

Перегрузка (Overloading): В Oracle PL/SQL можно создать несколько процедур с одним именем, но разными параметрами (разное количество или типы). В SQL Server перегрузка не поддерживается напрямую для процедур.

Oracle:

```
CREATE PROCEDURE process_data (p_id NUMBER) AS ...  
CREATE PROCEDURE process_data (p_name VARCHAR2) AS ...
```

## Предоставление и отзыв привилегий на исполнение:

```
GRANT EXECUTE ON имя_процедуры TO пользователь_роль;  
REVOKE EXECUTE ON имя_процедуры FROM пользователь_роль;
```

Типы данных (используются стандартные SQL-типы):

- Числовые: INT, DECIMAL, NUMBER, FLOAT.
- Строковые: VARCHAR(n), CHAR(n), CLOB.
- Дата/время: DATE, TIMESTAMP.
- Бинарные: BLOB, RAW.
- Специфические: BOOLEAN (PL/SQL), TABLE (T-SQL), CURSOR.

## Достоинства использования процедур:

- Производительность: Хранение в скомпилированном виде.
- Безопасность: Инкапсуляция логики, контроль доступа через EXECUTE.
- Целостность данных: Централизация логики, уменьшение ошибок.
- Повторное использование: Один раз созданная процедура может вызываться многократно.

## Недостатки:

- Сложность отладки: Требуется специализированные инструменты.
- Привязка к СУБД: Перенос между разными СУБД затруднен.
- Масштабируемость: Могут создавать нагрузку на сервер БД, если логика слишком сложная.

#### Процедуры PL/SQL (Oracle)

- Особенности: Встроенный язык Oracle, поддерживает объектно-ориентированные возможности, перегрузку, исключения, курсоры.
- Структура: Состоит из секций DECLARE, BEGIN, EXCEPTION, END.
- Пакеты: Группировка процедур, функций, переменных в единый модуль.

#### Особенности работы в Oracle / SQL Server

Аспект	Oracle (PL/SQL)	SQL Server (T-SQL)
Обработка ошибок	EXCEPTION блок (предопределенные и пользовательские исключения)	TRY...CATCH блок
Возврат результатов	Через OUT параметры или курсор (SYS_REFCURSOR)	OUTPUT параметры, RETURN, результирующие наборы (SELECT)
Транзакции	Управление внутри процедуры (COMMIT/ROLLBACK)	Автокоммит по умолчанию, но можно управлять явно
Отладка	Инструменты: SQL Developer, TOAD, DBMS_OUTPUT	SQL Server Management Studio (SSMS), PRINT
Временные таблицы	Глобальные временные таблицы (GTT)	Локальные(#) глобальные (##) временные таблицы

#### Системные процедуры:

- Oracle: DBMS\_OUTPUT.PUT\_LINE (вывод), DBMS\_SQL (динамический SQL), DBMS\_JOB (задания).
- SQL Server: sp\_help (справка об объекте), sp\_who (информация о процессах), sp\_rename (переименование объекта).
- MySQL: mysql.proc (метаданные процедур), SHOW CREATE PROCEDURE.

### 16. SQL/PSM. Хранимые функции: создание, удаление, изменение, объявление параметров, вызов функций. Детерминизм функций. Встроенные функции. Табличные функции и особенности их реализации в Oracle/SQL Server. Внешние функции: синтаксис, стили параметров. Внешние функции на C++ в Oracle.

#### Синтаксис создания функции в стандарте SQL/PSM и основных СУБД:

```

CREATE FUNCTION имя_функции ([список_параметров])
RETURNS тип_возвращаемого_значения
[DETERMINISTIC | NOT DETERMINISTIC]
[LANGUAGE SQL]
BEGIN
    -- тело функции
    RETURN значение;
END;
```

Удаление функции: DROP FUNCTION [IF EXISTS] имя\_функции;

Изменение функции: в большинстве СУБД функция заменяется новым определением через CREATE OR REPLACE FUNCTION:

Объявление параметров: параметры могут быть входными (IN), выходными (OUT), входно-выходными (INOUT).

```
CREATE FUNCTION calc_tax(IN amt DECIMAL, IN rate DECIMAL)
RETURNS DECIMAL ...
```

Вызов функций

- SELECT get\_employee\_salary(100) FROM dual;
- DECLARE sal DECIMAL(10,2); SET sal = get\_employee\_salary(100);

Детерминизм функций

- DETERMINISTIC: функция всегда возвращает одинаковый результат для одинаковых входных параметров.
- NOT DETERMINISTIC: результат может различаться при одинаковых параметрах (например, функция CURRENT\_TIMESTAMP).

Встроенные функции: Это предопределённые системные функции (UPPER(), ROUND(), CONCAT()).

Табличные функции возвращают таблицу (набор строк).

Oracle (PL/SQL): Используется RETURN с типом TABLE.

```
CREATE FUNCTION get_employees_by_dept(dept_id INT)
RETURN employees_table_type PIPELINED AS
BEGIN
    FOR rec IN (SELECT * FROM employees
                WHERE department_id = dept_id) LOOP
        PIPE ROW(rec);
    END LOOP;
    RETURN;
END;
```

Вызов: SELECT \* FROM TABLE(get\_employees\_by\_dept(10));

SQL Server (T-SQL): Используется RETURNS TABLE.

```
CREATE FUNCTION get_employees_by_dept(@dept_id INT)
RETURNS TABLE
AS
RETURN (SELECT * FROM employees
        WHERE department_id = @dept_id);
```

Вызов: SELECT \* FROM get\_employees\_by\_dept(10);

Внешние функции: функции, написанные на внешних языках (C, C++, Java).

Общий синтаксис (Oracle):

```
CREATE FUNCTION внешняя_функция(параметры)
RETURN тип
AS LANGUAGE C
LIBRARY имя_библиотеки
NAME "C_функция";
```

Стили параметров:

- BY VALUE – передача по значению.
- BY REFERENCE – передача по ссылке.

Внешние функции на C++ в Oracle:

```
cpp
extern "C" double calculate_tax(double income) {
    return income * 0.13;
}
```

**17. SQL/PSM. Триггеры: создание и удаление, триггерные таблицы, их назначение и особенности в SQL Server и Oracle. Вложение триггеров, порядок срабатывания, отключение триггеров. Дополнительные виды триггеров в SQL Server (триггер отката, рекурсии) и Oracle (Instead Of, строчный триггер).**

Триггер — это хранимая процедура, автоматически выполняемая при возникновении определенного события с таблицей или представлением.

SQL Server:

```
CREATE TRIGGER trigger_name
ON table_name
AFTER INSERT -- или INSTEAD OF, BEFORE (для DDL)
AS
BEGIN
    -- логика триггера
END;
```

Oracle:

```
CREATE TRIGGER trigger_name
BEFORE INSERT OR UPDATE ON table_name
FOR EACH ROW -- для строчных триггеров
BEGIN
    -- логика триггера
END;
```

Удаление триггера SQL Server/Oracle: DROP TRIGGER trigger\_name;

Триггерные таблицы (псевдотаблицы) используются для доступа к данным, затронутым операцией.

- INSERTED (SQLServer) / :NEW (Oracle): хранит новые данные.
- DELETED (SQL Server) / :OLD (Oracle): хранит старые данные.

Особенности:

- SQL Server: INSERTED и DELETED доступны только внутри триггера.
- Oracle: :NEW и :OLD доступны только в строчных триггерах.

Вложение триггеров и порядок срабатывания

Вложение триггеров: вызов одного триггера из другого.

SQL Server: вложенность 32 уровня. Управление через RECURSIVE\_TRIGGERS.

Oracle: вложенность возможна через вызовы процедур.

Порядок срабатывания:

SQL Server:

- INSTEAD OF триггеры.
- Прямые DML-действия.
- AFTER триггеры.

Oracle:

- BEFORE STATEMENT.
- BEFORE EACH ROW.
- DML-операция.
- AFTER EACH ROW.
- AFTER STATEMENT.

Отключение триггеров

SQL Server:

```
DISABLE TRIGGER trigger_name ON table_name;  
ALTER TABLE table_name DISABLE TRIGGER ALL;
```

Oracle:

```
ALTER TRIGGER trigger_name DISABLE;  
ALTER TABLE table_name DISABLE ALL TRIGGERS;
```

Дополнительные виды триггеров

SQL Server:

- Триггер отката: если в триггере выполняется ROLLBACK TRANSACTION, операция отменяется, и изменения не применяются.
- Рекурсивные триггеры: если RECURSIVE\_TRIGGERS = ON, триггер может вызвать сам себя.

Oracle:

- INSTEAD OF триггеры: выполняются вместо DML-операции, часто используются для представлений.
- Строчные триггеры (row-level): выполняются для каждой затронутой строки. Используют FOR EACH ROW и доступ к :NEW/:OLD.

## **18. Пакеты в Oracle. Хранимые модули. Внешние подпрограммы: стили параметров. Внешние подпрограммы в Interbase. Создание и вызов внешних подпрограмм в Oracle. Оценка размера БД. Проектирование журнала транзакций.**

Пакет — это объект БД Oracle, который инкапсулирует хранимые процедуры, функции, переменные, константы, курсоры и исключения. Состоит из:

- Спецификация — публичный интерфейс (заголовки процедур, функции, объявления переменных).
- Тело — реализация спецификации (код процедур, функций, приватные объекты).

К хранимым модулям в СУБД относятся:

- Хранимые процедуры — выполняют действия, не возвращают значение.
- Функции — возвращают значение.
- Триггеры — автоматически выполняются при событиях DML/DDDL.
- Пакеты — наборы процедур и функций.

Внешние подпрограммы: стили параметров

Внешние подпрограммы — программы, написанные на языках (C, Java и др.), вызываемые из БД. Стили передачи параметров:

- By Value — передача значения (копии).
- By Reference — передача адреса переменной (для изменения в программе).
- В Oracle для внешних процедур используется CALL-интерфейс, параметры передаются по ссылке.

В Interbase/Firebird внешние функции называются UDF (User-Defined Functions):

- Реализуются на языке C/C++ и компилируются в разделяемые библиотеки.
- Регистрируются в БД с помощью DECLARE EXTERNAL FUNCTION.
- Могут быть скалярными или возвращать несколько значений (с использованием указателей).
- Вызываются в SQL-запросах как обычные функции.

Создание и вызов внешних подпрограмм в Oracle

Этапы создания:

Написать внешнюю процедуру на каком-либо ЯП

Скомпилировать в разделяемую библиотеку (extlib.so или .dll)

Создать библиотечный объект в Oracle

Объявить процедуру в Oracle

Оценка размера БД выполняется для планирования ресурсов.

Основные шаги:

- Оценка размера таблиц:  $\text{Размер} = (\text{Строк}) \times (\text{Сумма размеров столбцов} + \text{Заголовок строки Oracle} (\sim 3\text{-}10 \text{ байт}))$ .
- Учёт индексов:  $\text{Размер индекса} \approx (\text{Строк}) \times (\text{Размер ключевых столбцов} + \text{Заголовок записи})$ .
- Дополнительные объекты: LOB-данные, временные таблицы, откат (UNDO).

Инструменты: Oracle Enterprise Manager, скрипты с использованием словаря данных (DBA\_SEGMENTS, DBA\_TABLES).

Журнал транзакций обеспечивает восстановление данных после сбоев.

Принципы проектирования:

- Размер файлов журнала:
  - Должен быть таким, чтобы копии не создавались слишком часто.
  - Ориентир: размер самого большого транзакционного пакета + 20%.
- Количество групп журнала:
  - Минимум 2 группы (рекомендуется 3-5).
  - Равный размер всех групп.
- Размещение:
  - Размещать на разных физических дисках.
- Режим работы:
  - ARCHIVELOG — для восстановления на момент сбоя.
  - NOARCHIVELOG — только для тестовых БД.
- Мониторинг:
  - Контролировать переключения журналов (V\$LOG\_HISTORY), избегать частых переключений (более 3-5 в час).

## 2. SQL/CLI, MDAC, XML, LINQ

**1. Интерфейс уровня вызовов. История. Структура интерфейса ODBC и основной алгоритм использования ODBC в прикладных программах**  
**Уровни соответствия ODBC (DDL, DML, выражения, типы данных). Назначение утилиты OdbcAdm.Exe. Методика создания DSN. Хранение сведений об источниках данных и драйверах ODBC. Варианты создания и использования строк подключения ODBC.**

Интерфейс уровня вызовов (Call Level Interface) — это стандартизированный программный интерфейс для доступа к СУБД, основанный на прямых вызовах функций из прикладного кода.

История: Разработан SQL Access Group в начале 1990-х. В 1993 году Microsoft выпустила первую версию ODBC — реализацию CLI для своей платформы, основанную на стандартах SAG и позже стандартизированную ISO/IEC как CLI.

Структура ODBC:

- Приложение — программа, использующая ODBC API.
- Диспетчер драйверов — библиотека (odbc32.dll), загружающая драйверы, проверяющая параметры, routing вызовов.
- Драйвер ODBC — библиотека, преобразующая ODBC-вызовы в вызовы конкретной СУБД.
- Источник данных — целевая СУБД/файл с данными.

Основной алгоритм использования ODBC в приложении:

- Инициализация: SQLAllocHandle.
- Установка атрибутов: SQLSetEnvAttr, SQLSetConnectAttr.
- Подключение: SQLConnect или SQLDriverConnect.
- Подготовка и выполнение запросов:
  - SQLAllocHandle для создания дескриптора оператора.
  - SQLPrepare / SQLExecute (с параметрами) или SQLExecDirect.
  - Привязка параметров (SQLBindParameter) и результатов (SQLBindCol).
- Выборка данных: SQLFetch или SQLFetchScroll в цикле.

- Завершение:
  - SQLFreeHandle для дескриптора оператора.
  - SQLDisconnect.
  - SQLFreeHandle для дескрипторов соединения и среды.

Уровни соответствия ODBC определяют, какие функции и возможности СУБД/драйвер поддерживает.

- Уровень соответствия Core (базовый): Базовый набор функций (подключение, выполнение SQL, выборка).
- Уровень 1: + Транзакции, каталоги, параметры в WHERE.
- Ур 2: + Прокручиваемые курсоры, массивы параметров, поиск по шаблону.

Соответствие по областям:

- DDL: Создание/удаление таблиц, индексов, представлений (уровень 2).
- DML: SELECT, INSERT, UPDATE, DELETE (Core).
- Выражения: Подзапросы, агрегатные функции, JOIN (уровень 1/2).
- Типы данных: Расширенные типы (дата/время, двоичные, длинные тексты).

OdbcAdm.exe (или odbcad32.exe) — администратор источников данных ODBC. Позволяет:

- Создавать, настраивать, удалять DSN.
- Тестировать подключения.
- Просматривать и управлять установленными драйверами ODBC.

DSN (Data Source Name) — именованная конфигурация для подключения к источнику данных.

Шаги создания через администратор ODBC:

Запустить odbcad32.exe.

- Выбрать вкладку Пользовательский DSN или Системный DSN.
- Нажать «Добавить».
- Выбрать драйвер для целевой СУБД (например, «SQL Server»).
- Ввести:
  - Имя DSN.
  - Сервер, базу данных, аутентификацию.
- Протестировать подключение и сохранить.

Хранение сведений об источниках данных и драйверах ODBC

- DSN хранятся в реестре Windows:
  - Пользовательские DSN: CURRENT\_USER\Software\ODBC\ODBC.INI
  - Системные DSN: LOCAL\_MACHINE\Software\ODBC\ODBC.INI
  - Файловые DSN — в текстовых файлах (.dsn)
- Драйверы ODBC: параметры хранятся в LOCAL\_MACHINE\Software\ODBC\ODBCINST.INI

Варианты создания и использования строк подключения ODBC:

- С использованием DSN: DSN=MyDSN;UID=user;PWD=password;
- DSN-less подключение (без предварительно созданного DSN): Driver={SQL Server};Server=server\_name;Database=db\_name;Trusted\_Connection=yes;
- Файловый DSN: "FILEDSN=C:\my.dsn;UID=user;PWD=password;"

- Динамическое построение в коде приложения: передача строки в SQLDriverConnect. Параметры строки: DRIVER, SERVER, DATABASE, UID, PWD, Trusted\_Connection, PORT и др.

## **2. Структурная схема доступа к БД через ODBC. Типы данных ODBC и их преобразование драйверами ODBC. Коды возврата ODBC и состояние ошибок. Функция идентификации ошибок и её назначение. Преобразование типов данных. Управление пулами соединений с помощью OdbcAdm.Exe, реестра и функций ODBC.**

Схема:

- Приложение – использует API ODBC.
- Диспетчер драйверов – загружает драйверы, управляет соединениями, обрабатывает вызовы ODBC.
- Драйвер ODBC – преобразует вызовы ODBC в команды СУБД.
- Источник данных – целевая БД.

Основные категории типов данных ODBC:

- SQL типы – типы, определяемые в SQL.
- C типы – типы данных в языке C.

Преобразование:

- Драйвер ODBC выполняет преобразование между SQL-типами и C-типами.
- Изменение формата (например, дата/время).
- Кодировки символов.
- Обработка значений NULL.

Коды возврата функций ODBC:

- SQL\_SUCCESS – успешное выполнение.
- SQL\_SUCCESS\_WITH\_INFO – успех, но с предупреждением.
- SQL\_NO\_DATA – данные отсутствуют.
- SQL\_ERROR – ошибка выполнения.
- SQL\_INVALID\_HANDLE – неверный дескриптор.

Состояния ошибок (SQLSTATE):

- 5-символьный код, детализирующий ошибку (например, HY000 – общая ошибка, 42000 – синтаксическая ошибка).
- Формат: Class (2 символа) + Subclass (3 символа).

Функция идентификации ошибок и её назначение: SQLError() или SQLGetDiagRec() / SQLGetDiagField().

- Назначение: Получение детальной диагностической информации об ошибках и предупреждениях.
- Возвращает:
  - SQLSTATE.
  - Нативный код ошибки СУБД.
  - Текстовое описание ошибки.
- Может вызываться для различных дескрипторов.

Преобразование типов данных:

- Явное преобразование – с использованием функций CAST/CONVERT.
- Неявное преобразование – автоматически драйвером ODBC при передаче данных между приложением и БД.
- Контроль преобразования – задание типов при связывании параметров и результатов (SQLBindParameter, SQLBindCol).

Управление пулами соединений

Использование OdbcAdm.Exe:

- Утилита администрирования ODBC (в Windows).
- Позволяет настраивать DSN (Data Source Name).
- Управление пулами: Вкладка "Пулы соединений" → установка времени ожидания, включение/выключение пулинга.

Через реестр (Windows):

- Ключ:  
HKEY\_LOCAL\_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\ODBC  
Connection Pooling.
- Параметры:
  - CPOutetimeout – время жизни соединения в пуле (в секундах).
  - Retry Wait – время ожидания при ошибке.

Функции ODBC:

- SQLSetEnvAttr() – установка атрибутов пулинга:
- SQLConnectionPool() – управление пулами (в зависимости от версии ODBC).

**3. Основные группы функций: назначение и отмена идентификаторов окружения, соединения и операторов. Примеры. Функции соединения и отсоединения от источника данных, выполнения SQL-операторов. Определение и установка основных опций соединения. Непосредственное и подготавливаемое выполнение операторов. Примеры.**

В SQL/CLI работа строится через дескрипторы (handles) трёх основных типов:

- Идентификатор окружения (SQLHENV): описывает общее окружение для работы с СУБД, глобальные настройки.
  - Отмена (освобождение): SQLFreeHandle(SQL\_HANDLE\_ENV, henv).
- Идентификатор соединения (SQLHDBC): управляет одним соединением с источником данных (Data Source).
  - Отмена: SQLFreeHandle(SQL\_HANDLE\_DBC, hdbc) после отключения.
- Идентификатор оператора (SQLHSTMT): представляет SQL-запрос, его параметры, результаты.
  - Отмена: SQLFreeHandle(SQL\_HANDLE\_STMT, hstmt).

SQLHENV henv;

SQLHDBC hdbc;

SQLHSTMT hstmt;

```

SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (void*)SQL_OV_ODBC3,
0);
SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
// ... соединение с БД ...
SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
// После использования:
SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
SQLDisconnect(hdbc);
SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
SQLFreeHandle(SQL_HANDLE_ENV, henv);

```

Функции соединения и отсоединения от источника данных

- SQLConnect(hdbc, "DSNname", SQL\_NTS, "user", SQL\_NTS, "password", SQL\_NTS) или SQLDriverConnect
- SQLDisconnect(hdbc) — закрывает соединение, но оставляет дескриптор соединения для возможного переподключения.

Непосредственное выполнение SQL-операторов: оператор выполняется сразу при вызове функции. Функция: SQLExecDirect(hstmt, sql\_string, SQL\_NTS).

Подготавливаемое выполнение:

- Подготовка: SQLPrepare(hstmt, sql\_string, SQL\_NTS) — синтаксический разбор и создание плана.
- Возможная привязка параметров через SQLBindParameter.
- Выполнение: SQLExecute(hstmt) — можно выполнять многократно с разными параметрами.

Определение и установка основных опций соединения:

Установка атрибутов выполняется через SQLSetConnectAttr.

Примеры атрибутов соединения:

- SQLSetConnectAttr(hdbc, SQL\_ATTR\_AUTOCOMMIT, (void\*)SQL\_AUTOCOMMIT\_OFF, 0).
- SQL\_ATTR\_LOGIN\_TIMEOUT: таймаут подключения в секундах.
- SQL\_ATTR\_CURRENT\_CATALOG: установка текущей базы данных.

**4. Функции получение результатов, назначения области памяти и типа данных для столбца таблицы, определение числа строк и столбцов, выборки данных. Определение и работа с курсорами. Примеры. Обзор дополнительных функций ODBC. Получение данных о драйверах, типах данных, основных опциях драйвера, данных о таблицах.**

После выполнения SQL-запроса приложение работает с результатами через следующие ключевые функции:

- SQLFetch/SQLFetchScroll: Получение строк(и) данных из результирующего набора.
- SQLBindCol: Эта функция связывает столбец результата набора переменной в приложении. Указывается:
  - Номер столбца.
  - Тип данных C, к которому будет преобразовано значение.

- Адрес буфера для данных.
- Адрес буфера для индикатора длины.
- SQLGetData: Выборка данных из непривязанного столбца..
- SQLNumResultCols: Определение числа столбцов в результирующем наборе.
- SQLRowCount: Определение числа строк, затронутых оператором.

В ODBC курсор – это программный объект, который представляет позицию в результирующем наборе и позволяет перемещаться по строкам.

Типы курсоров: Управляются через атрибуты инструкции (SQLSetStmtAttr).

Тип курсора: SQL\_ATTR\_CURSOR\_TYPE. Значения:

- SQL\_CURSOR\_FORWARD\_ONLY (только вперёд, по умолчанию)
- SQL\_CURSOR\_STATIC (статический снимок)
- SQL\_CURSOR\_KEYSET\_DRIVEN (управляемый набором ключей)
- SQL\_CURSOR\_DYNAMIC (динамический)

Обзор дополнительных функций ODBC для получения метаданных

Получение данных о драйверах и источниках данных:

- SQLDrivers: Список установленных драйверов и их атрибутов.
- SQLDataSources: Список доступных источников данных (DSN).

Получение информации о возможностях драйвера и типах данных:

- SQLGetInfo: Возвращает основные опции и возможности драйвера и СУБД.
- SQLGetTypeInfo: Возвращает список типов данных SQL, поддерживаемых источником данных, с подробным описанием (имя, точность, nullable и т.д.).

Получение данных о схеме базы данных (таблицах, столбцах):

- SQLTables: Возвращает список таблиц, каталогов или схем. Можно фильтровать по имени, типу ("TABLE", "VIEW", "SYSTEM TABLE").
- SQLColumns: Возвращает список столбцов для указанной таблицы.
- SQLPrimaryKeys: Возвращает информацию о первичных ключах таблицы.
- SQLForeignKeys: Возвращает информацию о внешних ключах.
- SQLStatistics: Возвращает статистику и индексы для таблицы.

**5. SQL-расширения ODBC. Процедуры, скалярные функции. Управления транзакциями. Уровни изоляции. Исходный SQL. SQL-расширения ODBC. Внешние объединения, дополнительные типы данных. Скалярные функции. Предикат Escape.**

ODBC — это стандарт CLI, который для обеспечения совместимости с различными СУБД использует несколько уровней SQL:

- Исходный SQL (SQL-92) — это форма, которую драйвер и менеджер драйверов передают СУБД.
- Расширения ODBC — это синтаксис, выходящий за рамки SQL-92, который ODBC либо преобразует в "Исходный SQL", либо передает напрямую СУБД, если она его поддерживает. К ним относятся:
  - Скалярные функции (см. ниже).
  - Предикат ESCAPE (см. ниже).
  - Внешние объединения (см. ниже).

- Специальные посл-ти для дат, времени/ escape-последовательности.

Внешние объединения: SQL-92 определяет синтаксис LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN. Однако старые СУБД использовали собственный синтаксис.

Расширение ODBC: Для совместимости ODBC вводит специальный синтаксис в предложении WHERE с использованием escape-последовательности.

Стандарт SQL-92:

```
SELECT * FROM Customers LEFT OUTER JOIN Orders ON Customers.ID = Orders.CustomerID
```

Расширение ODBC:

```
SELECT * FROM {oj Customers LEFT OUTER JOIN Orders ON Customers.ID = Orders.CustomerID}
```

Дополнительные типы данных (C и SQL)

ODBC определяет собственные типы данных для C и SQL, чтобы абстрагироваться от различий в СУБД.

- Типы данных C: Используются в буферах приложения (SQL\_C\_CHAR, SQL\_C\_LONG, SQL\_C\_DOUBLE, SQL\_C\_DATE, SQL\_C\_BINARY).
- Типы данных SQL: Используются для описания типов столбцов в базе данных (SQL\_CHAR, SQL\_INTEGER, SQL\_DATE, SQL\_BLOB).

Скалярные функции: это расширение ODBC, позволяющее использовать в запросах стандартизированные функции для работы со строками, датами, числами, типами данных и системой.

Примеры:

- Строковые: CONCAT(строка1, строка2), SUBSTRING(строка, начало, длина), UPPER(строка)
- Числовые: ABS(число), ROUND(число, точность), LOG(число)
- Дата/время: CURRENT\_DATE, YEAR(дата), MONTH(дата)
- Системные: USER(), DATABASE()

Использование в запросе: Для вызова скалярной функции используется escape-последовательность {fn ... }: SELECT {fn CONCAT(LastName, {fn SPACE(2)}}, FirstName)} FROM Employees

Предикат ESCAPE (для LIKE) Используется для поиска символов '%', '\_' и самого escape-символа в операторе LIKE. Стандартный SQL требует указания escape-символа. Расширение ODBC: Позволяет задать escape-символ с помощью специальной конструкции {escape ... } в конце предиката LIKE.

SQL: SELECT \* FROM Products WHERE Name LIKE '%50\%%%' ESCAPE '\'

ODBC: SELECT \* FROM Products WHERE Name LIKE '%50\%%%' {escape '\}'

Процедуры и управление транзакциями. Уровни изоляции.

Хранимые процедуры: Для вызова ODBC предоставляет {call ... }.

- Без параметров: {call GetTopCustomers}
- С параметрами: {call UpdateSalary(?, ?)}.
- С возвратом значения: {? = call CalculateBonus(?)}

Управление транзакциями:

По умолчанию ODBC работает в режиме auto-commit.

Для явного управления транзакциями необходимо:

- Выключить auto-commit:
  - SQLSetConnectAttr(ConnectionHandle, SQL\_ATTR\_AUTOCOMMIT, SQL\_AUTOCOMMIT\_OFF, 0);
- Выполнять SQL-запросы.
- Фиксировать изменения:
  - SQLEndTran(SQL\_HANDLE\_DBC, ConnectionHandle, SQL\_COMMIT);
- Или откатывать:
  - SQLEndTran(SQL\_HANDLE\_DBC, ConnectionHandle, SQL\_ROLLBACK);

Уровень изоляции устанавливается на соединении (атрибут SQL\_ATTR\_TXN\_ISOLATION) и определяет степень видимости изменений параллельных транзакций.

Уровни ODBC (от низкой к высокой изоляции):

- SQL\_TXN\_READ\_UNCOMMITTED: "Грязное" чтение.
- SQL\_TXN\_READ\_COMMITTED: Защита от "грязного" чтения.
- SQL\_TXN\_REPEATABLE\_READ: Защита от "грязного" и неповторяющегося чтения.
- SQL\_TXN\_SERIALIZABLE: Полная изоляция.
- SQL\_TXN\_SNAPSHOT: Уровень, связанный с управлением версиями строк.

Установка: SQLSetConnectAttr(ConnectionHandle, SQL\_ATTR\_TXN\_ISOLATION, (SQLPOINTER)SQL\_TXN\_READ\_COMMITTED, 0);

**6. BDE/FireDAC: структурная схема доступа к БД, хранение сведений о драйверах и источниках данных, пулы соединений. OCI. Алгоритм подключения к Oracle. Достоинства и недостатки. MDAC. OLE DB: интерфейсы базового уровня, структурная схема доступа. Создание строки подключения, хранение настроек. Провайдеры. Пулы соединений OLE DB. Настройка параметров пулов через реестр и OLE DB API.**

Структурная схема доступа: Приложение → BDE/FireDAC API → Драйвер (для конкретной СУБД) → Сетевая библиотека → СУБД.

Хранение сведений: BDE использует файлы IDAPI.CFG и реестр Windows для хранения конфигураций драйверов (например, Paradox, dBASE) и источников данных (Aliases). FireDAC хранит параметры подключения в файлах .ini, реестре или в коде.

Пулы соединений: BDE поддерживает ограниченный пул через SHAREDMEMORYLOCATION. FireDAC имеет встроенную систему пулинга с настройками времени жизни, размера пула и валидации соединений.

OCI (Oracle Call Interface):

Алгоритм подключения к Oracle:

- Загрузка клиентской библиотеки OCI (oci.dll).
- Инициализация среды OCI.
- Создание контекста подключения (handle).
- Установка атрибутов подключения (хост, порт, SID/Service Name).
- Вызов OCIServerAttach и OCISessionBegin.

Плюсы: Высокая производительность, полная поддержка возможностей Oracle.

Минусы: Требуется установка клиента Oracle, сложность API.

MDAC (Microsoft Data Access Components): Набор библиотек (OLE DB, ODBC, ADO) для унифицированного доступа к данным.

Компоненты:

- OLE DB – низкоуровневый доступ на основе COM.
- ODBC – интерфейс для реляционных СУБД.
- ADO – высокоуровневая объектная модель поверх OLE DB.

OLE DB (Object Linking and Embedding for Databases)

Интерфейсы базового уровня:

- IDBInitialize – инициализация источника данных.
- IDBCreateSession – создание сессии для работы с командами.
- IOpenRowset – открытие набора строк (аналог таблицы).
- IDBCreateCommand – создание команд (если провайдер поддерживает).
- ICommandText – выполнение SQL-запросов.

Структурная схема доступа:

Приложение → ADO/OLE DB API → OLE DB Provider (SQLOLEDB) → СУБД

Провайдер – COM-объект, преобразующий вызовы OLE DB в вызовы конкретной СУБД.

Строка подключения содержит параметры:

- Provider: имя OLE DB-провайдера (SQLOLEDB.1).
- Data Source: сервер/экземпляр СУБД.
- Initial Catalog: база данных.
- Authentication: Integrated Security=SSPI или User ID/Password.

"Provider=SQLOLEDB.1;Data Source=ServerName;Initial Catalog=DBName;Integrated Security=SSPI"

Хранение настроек:

- В файле конфигурации приложения (.ini, .config, .xml).
- В реестре Windows (HKEY\_CURRENT\_USER\Software...).
- В коде приложения (возможно, с шифрованием).

## Пулы соединений OLE DB

Настройка через реестр:

Ключ: HKEY\_CLASSES\_ROOT\CLSID\{<CLSID провайдера>}\SPOutput  
SPOutput – время жизни соединения в пуле (в секундах).

Настройка через OLE DB API:

Через свойства DBPROPSET\_DBINIT:

- DBPROP\_INIT\_TIMEOUT – таймаут инициализации.
- DBPROP\_INIT\_OLEDBSERVICES – управление пулингом.

Важные параметры пула:

- Retry Wait – время ожидания перед повторной попыткой.
- Expiration Timeout – время неактивности до закрытия соединения.
- Max Pool Size – максимальное число соединений в пуле.

**7. Модель объектов ADO. Структурная схема доступа. Создание соединения и хранение настроек. Недостатки ADO. Библиотека MFC. Методика создания и особенности приложений с MFC для ODBC (с помощью мастера и без мастера). Достоинства и недостатки. Библиотека ATL. Достоинства и недостатки. Пулы соединений OLE DB.**

Модель объектов ADO (ActiveX Data Objects) – это высокоуровневая COM-библиотека для доступа к данным, построенная поверх OLE DB. Основные объекты:

- Connection: даёт соединение с источником данных, управляет транзакциями.
- Command: Определяет команду для выполнения.
- Recordset: Набор записей, полученных в результате выполнения команды.
- Record: Представляет одну строку данных или иерархическую структуру.
- Stream: Представляет поток двоичных или текстовых данных.
- Field: Представляет один столбец в Recordset.
- Parameter: Представляет параметр команды Command.
- Property: Динамические характеристики, предоставляемые провайдером.

Структурная схема доступа (ADO):

Клиентское приложение → ADO → Провайдер OLE DB → Источник данных (SQL Server, Oracle, файл и т.д.).

Создание соединения: Экземпляр объекта Connection создается и настраивается строкой соединения (ConnectionString), после чего вызывается метод Open().

Хранение настроек: Строка подключения часто хранится:

- В конфигурационных файлах приложения.
- В системном реестре Windows.
- В DSN (Data Source Name) – системных или пользовательских.
- В UDL (Universal Data Link) файлах.
- В среде разработки (как свойство проекта или в ресурсах).

Недостатки ADO.

- COM-зависимость: Требуется наличия и регистрации COM-компонентов.
- Отсутствие строгой типизации.
- Сложность с отключенными данными.
- Ресурсоемкость: Объекты требуют явного освобождения.

MFC (Microsoft Foundation Classes) – каркас приложений на C++ для Windows, предоставляющий классы-обертки для API Windows, включая доступ к данным через ODBC.

Методика создания с помощью Мастера (AppWizard):

- Создание проекта MFC Application (например, Single Document).
- На шаге Database Support выбрать Database view without file support и нажать Data Source....
- Выбрать тип ODBC, указать DSN, таблицу или запрос.
- Мастер сгенерирует:
  - Класс приложения, производный от CWinApp.
  - Класс CRecordset-производный для работы с набором данных.
  - Класс CRecordView-производный для формы, отображающей данные.
  - Класс документа (может не использоваться для данных).
- Связь между CRecordView и CRecordset осуществляется через RFX (Record Field Exchange) механизм.

Особенности приложений с MFC для ODBC:

- Класс CDatabase: Объект соединения с БД.
- Класс CRecordset: Представляет набор записей. Содержит RFX-вызовы для обмена данными с полями БД.
- Класс CRecordView: Форма (диалог) с элементами управления, привязанными к полям CRecordset.
- Мастер генерирует каркас приложения с базовой навигацией (First, Last, Next, Prev) и возможностью редактирования.

Создание без Мастера:

- Ручное наследование от CDatabase, CRecordset, CFormView/CDialog.
- Реализация RFX в методе DoFieldExchange() производного класса CRecordset.
- Самостоятельное создание элементов управления и привязка их данных через DDX\_Field\* функции в методе DoDataExchange() формы.

Достоинства MFC+ODBC:

- Быстрая разработка типовых приложений "форма-БД" с помощью мастера.
- Естественная интеграция в C++/Windows-среду.
- Прямой доступ к мощному GUI API MFC.

Недостатки MFC+ODBC:

- Привязка к платформе Windows.
- Сложность и громоздкость кода MFC для новичков.
- Ориентация на связанную архитектуру.

ATL (Active Template Library) – библиотека шаблонов C++ для создания компактных и быстрых COM-компонентов.

Достоинства ATL:

- Высокая производительность: Шаблонный код инлайнится компилятором, накладные расходы минимальны. Прямой доступ к OLE DB, минуя ODBC.
- Компактность: Генерирует минимальный по размеру код.
- Гибкость: Предоставляет низкоуровневый контроль над доступом к данным.
- Идеально для создания COM-серверов, включая поставщиков и потребителей OLE DB.

Недостатки ATL:

- Сложность разработки: Требуется понимания COM, OLE DB, шаблонов C++.
- Отсутствие автоматизированных мастеров для быстрого создания форм.
- Меньше готовых компонентов для создания пользовательского интерфейса.
- Сложность отладки шаблонного кода.

Пул соединений OLE DB – механизм, позволяющий повторно использовать физические соединения с источником данных между различными клиентскими сеансами.

Принцип работы:

- Клиент запрашивает соединение с определенными параметрами (строка подключения).
- Драйвер/провайдер проверяет пул на наличие свободного, но неактивного соединения с такими же параметрами.
- Если такое соединение найдено, оно возвращается клиенту. Если нет – создается новое.
- После закрытия соединения клиентом (Release или Close) оно не уничтожается, а возвращается в пул в состояние бездействия.

В OLE DB за пулы соединений отвечает компонент Resource Pooling Service. Ее можно включить или отключить в строке подключения.

**8. Недостатки пулов соединений OLE DB и ODBC. Организация пула соединений ADO. Параметры настройки в строке соединения. Поддержка пулов в .NET Data Provider. Особенности пулов в ADO.NET. Мониторинг за пулами соединений, фрагментация пулов. Особенности объектных пулов. Пулы в Oracle.**

Недостатки пулов соединений OLE DB и ODBC

- Сложность управления: Пулы в OLE DB и ODBC управляются на уровне драйверов/провайдеров и менеджера драйверов.
- Ограниченная диагностика.
- Фрагментация пулов: Ключевая проблема. Пул идентифицируется по строке соединения. Даже незначительное изменение в строке создает новый пул.
- Зависимость от реализации провайдера: Поведение и эффективность пула сильно зависят от качества реализации конкретного провайдера или драйвера.

В классическом ADO пул соединений обеспечивался не самой библиотекой ADO, а нижележащим OLE DB Service Component - службой Session Pooling: Включение/выключение и настройка пула выполнялись через свойства OLE DB провайдера в строке соединения или через системные механизмы.

В ADO.NET и современных провайдерах основные параметры пула задаются в строке подключения:

- Pooling=True/False.
- Max Pool Size / Min Pool Size.
- Connection Lifetime – Время жизни соединения в секундах.
- Connection Timeout – Время ожидания установления нового соединения.

Поддержка пулов в .NET Data Provider / Особенности пулов в ADO.NET

- Провайдер-специфичная реализация: Каждый .NET Data Provider реализует собственный механизм пула.
- Автоматическое управление: Пул включен по умолчанию.
- Требование к корректному закрытию: Критически важно явно закрывать соединение.
- Сброс состояния соединения: При возвращении в пул, ADO.NET выполняет "сброс" соединения. Если сброс не удастся, соединение уничтожается.

Мониторинг за пулами соединений, фрагментация пулов

Мониторинг:

- Счетчики производительности: ADO.NET и некоторые провайдеры предоставляют счетчики:
  - NumberOfActiveConnectionPools
  - NumberOfActiveConnections
  - NumberOfFreeConnections
- Динамические административные представления SQL Server: Запросы к sys.dm\_exec\_connections, sys.dm\_exec\_sessions.
- Профилировщики: SQL Server Profiler или расширенные события могут показывать события Audit Login и Audit Logout.

Фрагментация пулов: возникает, когда приложение создает множество пулов из-за незначительных отличий в строках подключения (разный порядок параметров, регистр, учетные записи, имена БД). Результат:

- Избыточное потребление ресурсов на клиенте и сервере.
- Не достигается эффективное повторное соединений.
- Меры борьбы: Использовать единый способ построения строки, хранить ее централизованно, использовать одинаковые учетные данные.

Объектный пул - паттерн для переиспользования любых объектов БД. Общий объектный пул должен управлять состоянием объектов, их инициализацией и валидацией.

Сложности: Очистка состояния объекта при возврате в пул, обработка "битых" объектов, потокобезопасность.

Oracle предоставляет несколько технологий пулинга соединений:

- Пул на стороне клиента: Аналогично в ADO.NET реализован провайдерами:
  - Oracle Data Provider for .NET: Использует параметры строки подключения Pooling, Min Pool Size, Max Pool Size, Connection Lifetime. Известен высокой эффективностью и богатыми возможностями мониторинга через OracleConnection.GetSessionInfo().
  - Oracle Provider for OLE DB: Использует механизмы пула OLE DB.
- Пул соединений на стороне сервера: Более продвинутая технология.
  - Принцип: брокер внутри экземпляра Oracle поддерживает пул серверных процессов. Клиентские приложения подключаются к брокеру, который выдает им свободную серверную сессию.
  - Использование: В строке подключения указывается SERVER=POOLED.

## **9. Особенности взаимодействия Java-программ и БД. Типы драйверов JDBC и их особенности. JDK 1.1. Назначение JRE, настройка путей доступа. Основные интерфейсы Java. Обзор основных классов JDBC. Сценарии использования JDBC: доступы к БД из апплетов, приложений. Классы для работы с метаданными и управления транзакциями. Oracle JDBC.**

Особенности взаимодействия Java-программ и БД: взаимодействие строится на основе стандарта JDBC, основанном на SQL API, который позволяет Java-коду:

- Устанавливать соединения с СУБД.
- Формировать и выполнять SQL-запросы.
- Обращивать результирующие наборы данных.
- Управлять транзакциями.

Ключевая особенность — абстракция от конкретной СУБД. Программа работает с интерфейсами JDBC, а драйвер, специфичный для базы данных, реализует эти интерфейсы.

Существует 4 типа драйверов JDBC:

- JDBC-ODBC мост (Bridge Driver): Преобразует вызовы JDBC в вызовы ODBC. Медленный, зависит от платформы.
- Частично-нативный API (Native-API Driver): Часть драйвера на Java, часть — нативные библиотеки клиента СУБД. Требуется установка клиентского ПО БД на машину с приложением. Быстрее типа 1, но менее портативен.
- Java для промежуточного сервера (Network Protocol Driver): Драйвер на Java, который преобразует вызовы JDBC в независимый от БД протокол, который затем промежуточный сервер преобразует в протокол конкретной СУБД. Не требует клиентских библиотек.
- Java, прямой доступ к БД (Thin Driver): Полностью написан на Java, общается с СУБД напрямую по сетевому протоколу. Полностью портабелен.

JDK 1.1: Версия Java Development Kit, в которой JDBC стал стандартным компонентом (пакет java.sql).

Назначение JRE (Java Runtime Environment): Среда для запуска уже скомпилированных Java-приложений. Содержит Java Virtual Machine и базовые библиотеки классов. Для запуска приложения, использующего JDBC, на клиентской машине достаточно JRE.

Настройка путей доступа (CLASSPATH): Для работы JDBC необходимо, чтобы JRE могла найти файлы JAR с драйвером БД. Путь к JAR-файлу драйвера должен быть добавлен в переменную среды CLASSPATH или указан при запуске приложения с помощью ключа -cp.

Основные интерфейсы Java (пакет java.sql и javax.sql)

- java.sql.Driver: Базовый интерфейс.
- java.sql.DriverManager (старый): Класс для управления набором драйверов, устаревший метод получения соединения (DriverManager.getConnection()).
- javax.sql.DataSource (новый): Предпочтительный фабричный интерфейс для получения соединений.
- java.sql.Connection: Представляет активное соединение с конкретной БД.
- java.sql.Statement, PreparedStatement, CallableStatement: Интерфейсы для выполнения статических SQL-запросов, параметризованных запросов и хранимых процедур соответственно.
- java.sql.ResultSet: Представляет таблицу данных, полученную в результате выполнения запроса.
- java.sql.ResultSetMetaData: Предоставляет информацию о структуре.
- java.sql.DatabaseMetaData: Предоставляет полную информацию о самой БД.

Обзор основных классов JDBC

- DriverManager: Старый класс регистрации драйверов и создания соединений.
- Types: Содержит константы для идентификации типов данных SQL.
- SQLException: Проверяемое исключение, которое выбрасывается при возникновении ошибок при работе с БД.
- Date, Time, Timestamp: Для работы с датой и временем в SQL-контексте.

Сценарии использования JDBC:

- Из standalone-приложений (Java SE):
  - Драйвер JAR подключается к приложению.
  - Используется DriverManager или DataSource для получения соединения.
  - Типы драйверов: Thin, Native API.
- Из веб-приложений (Java EE / Jakarta EE):
  - Драйвер JAR размещается на сервере приложений (например, Tomcat).
  - Настройка пула обычно осуществляется на уровне сервера приложений.
  - Приложение запрашивает DataSource через JNDI-контекст.
  - Типы драйверов: Thin.
- Из апплетов (устаревшая технология):
  - Апплет работает в "песочнице" браузера.
  - Для прямого доступа к БД с машины клиента требовался драйвер Middleware или Thin.
  - Типичный подход: апплет общался с сервлетом/серверным приложением (через HTTP/RMI), а тот уже через JDBC работал с БД.

Метаданные:

- DatabaseMetaData: Получается из Connection (conn.getMetaData()). Используется для исследования возможностей и схемы БД.
- ResultSetMetaData: Получается из ResultSet (rs.getMetaData()). Используется для динамической обработки результатов запроса.

Управление транзакциями осуществляется через методы интерфейса Connection:

- `conn.setAutoCommit(false);` — отключает автокоммит.
- `conn.commit();` — фиксирует все изменения.
- `conn.rollback();` — откатывает все изменения.
- `conn.setTransactionIsolation(int level);` — устанавливает уровень изоляции.

Oracle JDBC

Основной драйвер: `oracle.jdbc.OracleDriver`.

URL соединения (Thin-драйвер, Тип 4): `jdbc:oracle:thin:@[host]:[port]:[sid] / jdbc:oracle:thin:@//[host]:[port]/[service_name]`.

Дополнительные возможности:

- Поддержка расширенных типов данных Oracle.
- Собственные расширения производительности.

**10. Классы JDBC: `java.sql.DriverManager`, `java.sql.Connection`, `java.sql.ResultSet`, `java.sql.Statement`, `java.sql.PreparedStatement`, `java.sql.CallableStatement`, `java.sql.SQLException`, `java.sql.SQLWarning`, `java.sql.Date`, `java.sql.Time`, `java.sql.Timestamp`. Примеры использования. Пулы соединений JDBC и их особенности.**

`java.sql.DriverManager`: для соединения с СУБД через строку подключения.

```
Class.forName("com.mysql.cj.jdbc.Driver");
Connection conn = DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/testdb", "user", "password"
);
```

`java.sql.Connection`: для создания объектов `Statement`, управления транзакциями.

```
Connection conn = DriverManager.getConnection(...);
conn.setAutoCommit(false);
```

`java.sql.Statement`: для выполнения статических SQL-запросов без параметров.

```
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("SELECT * FROM employees");
while (rs.next()) {
    System.out.println(rs.getString("name"));
}
stmt.close();
```

`java.sql.PreparedStatement`: Для выполнения параметризованных SQL-запросов.

```
PreparedStatement pstmt = conn.prepareStatement("INSERT INTO users (id, name)
VALUES (?, ?)");
pstmt.setInt(1, 101);
pstmt.setString(2, "Alice");
pstmt.executeUpdate();
pstmt.close();
```

java.sql.CallableStatement: Для вызова хранимых процедур в БД.  
CallableStatement cstmt = conn.prepareCall("{call get\_employee(?, ?)}");  
cstmt.setInt(1, 1001);  
cstmt.registerOutParameter(2, Types.VARCHAR);  
cstmt.execute();  
String empName = cstmt.getString(2);  
cstmt.close();

java.sql.ResultSet: Представляет результат выполнения запроса.  
ResultSet rs = stmt.executeQuery("SELECT \* FROM products");  
while (rs.next()) {  
    int id = rs.getInt("id");  
    String name = rs.getString("name");  
    double price = rs.getDouble("price");  
}  
rs.close();

java.sql.SQLException: Исключение, выбрасываемое при ошибках работы с БД.  
try {  
    // JDBC операции  
} catch (SQLException e) {  
    System.out.println("Код ошибки: " + e.getErrorCode());  
    System.out.println("Сообщение: " + e.getMessage());  
}

java.sql.SQLWarning: Для получения предупреждений от БД.  
SQLWarning warning = conn.getWarnings();  
while (warning != null) {  
    System.out.println("Warning: " + warning.getMessage());  
    warning = warning.getNextWarning();  
}

java.sql.Date, java.sql.Time, java.sql.Timestamp: Классы для работы с датой, временем и временными метками.  
java.sql.Date sqlDate = new java.sql.Date(System.currentTimeMillis());  
PreparedStatement pstmt = conn.prepareStatement("INSERT INTO orders (date) VALUES (?)");  
pstmt.setDate(1, sqlDate);  
Timestamp timestamp = rs.getTimestamp("created\_at");

## 2. Пулы соединений JDBC и их особенности

Назначение пулов соединений: Управление набором предварительно установленных соединений с БД для повышения производительности и эффективности использования ресурсов.

Особенности:

- Эффективность: Избегание расходов на установление соединения.
- Управление ресурсами: Ограничение максимального числа соединений.
- Поддержка транзакций: Пул может управлять распределёнными транзакциями.

- Валидация соединений: Проверка активности соединения перед его выдачей клиенту.
- Возврат соединений: Клиент обязан явно закрывать соединение.

Ключевые параметры пулов:

- `initialSize` – начальное количество соединений.
- `maxTotal` – максимальное число активных соединений.
- `maxIdle` – максимальное число простаивающих соединений.
- `minIdle` – минимальное число простаивающих соединений.
- `maxWaitMillis` – время ожидания свободного соединения.
- `validationQuery` – SQL-запрос для валидации соединения.

**11. Особенности XML документов. Структура, схемы объявления и типы данных XML документов. Обзор XML БД и XQuery процессоров. Основные API для работы с XML БД. Спецификация XQJ и её реализация. Структурная схема доступа к XML БД. Драйверы. Примеры.**

XML (eXtensible Markup Language) — это язык разметки, предназначенный для хранения и передачи структурированных данных. Его основные особенности:

- Самоописываемость: Теги и структура документа определяет разработчик.
- Платформонезависимость: Может использоваться в любых ОС и средах.
- Человечитаемость: Данные представлены в текстовом виде.
- Иерархичность: Данные организованы в виде дерева.
- Расширяемость: Позволяет создавать пользовательские теги и атрибуты.
- Поддержка Unicode: Представление данных на разных языках.

## 2. Структура XML документа

Базовая структура включает:

xml

```
<?xml version="1.0" encoding="UTF-8"?> <!-- Объявление XML -->
<!-- Корневой элемент -->
<bookstore>
  <book category="IT">
    <title>XML Basics</title>
    <author>John Doe</author>
    <price>29.99</price>
  </book>
</bookstore>
```

- Объявление XML: Указывает версию и кодировку.
- Корневой элемент: Единственный элемент верхнего уровня.
- Элементы и атрибуты: Данные хранятся в элементах (тегах) и атрибутах.
- Пространства имён (Namespaces): Для избежания конфликтов имен.

Схемы объявления:

DTD (Document Type Definition):

- Определяет структуру и допустимые элементы/атрибуты.
- Ограниченные возможности типизации данных.
 

```
<!ELEMENT bookstore (book*)>
<!ELEMENT book (title, author, year, price)>
```

XML Schema (XSD): Более мощная альтернатива DTD.

- Поддержка типов данных, пространств имён.  
    <xs:element name="price" type="xs:decimal"/>  
    <xs:element name="year" type="xs:integer"/>

Типы данных в XML Schema:

- Простые типы: string, decimal, integer, date, boolean.
- Сложные типы: Структуры с вложенными элементами.
- Пользовательские типы: Ограничения через xs:restriction.

XML Базы данных:

- Нативные XML БД: Хранят XML в естественной иерархической форме.
- Гибридные БД: Реляционные БД с поддержкой XML.

XQuery: Язык запросов к XML данным (аналог SQL для XML).

Процессоры: BaseX, Saxon, Oracle XML DB.

```
for $book in doc("books.xml")/bookstore/book
where $book/price > 30
return $book/title
```

Основные API для работы с XML БД

- DOM: Загружает весь документ в память, позволяет менять структуру.
- SAX: Последовательное чтение, Экономит память.
- JAXP: Интеграция DOM/SAX в Java.
- XQJ: Стандартный API для выполнения XQuery запросов.

Спецификация XQJ (XQuery API for Java) и её реализация

Основные интерфейсы:

- XQDataSource: Источник данных.
- XQConnection: Соединение с XML БД.
- XQExpression: Выполнение запросов.
- XQResultSequence: Результат запроса.

Реализации:

- BaseX: org.basex.api.xqj.
- Saxon: net.sf.saxon.xqj.
- Oracle: oracle.xquery.xqj.

Структурная схема доступа к XML БД

Приложение Java → [XQJ API] → XQJ Драйвер → XML БД → Хранилище XML

Драйверы

- BaseX XQJ Driver: basex-xqj.jar.
- Saxon XQJ Driver: saxon-xqj.jar.
- Oracle XML DB XQJ Driver: Входит в состав СУБД Oracle.

**12. Язык XQuery. История создания. Назначение XPath. Особенности и основные группы типов данных. Типы данных величин узлов и выражений. Типизация, проверка и извлечение данных из узлов. Последовательности. Выражения: первичные, логические, пути (предикаты, сокращения), последовательности, арифметические, сравнения. Функции: особенности вызова и их обзор: группы и назначение.**

XQuery — язык запросов к XML-данным, разработанный W3C. Работа над ним началась в конце 1990-х годов на основе существующих проектов:

- Quilt (гибрид XPath, XML-QL, SQL) стал основой синтаксиса.
- XPath1.0(1999) — база для навигации по XML.
- XPath2.0(2007) и XQuery 1.0 (2007) стали рекомендациями W3C одновременно, разделяя общую модель данных и библиотеку функций.
- XQuery3.0(2014) добавил группировку, оконные функции, JSON-поддержку.

XPath (XML Path Language) — язык для адресации частей XML-документа, используемый в XQuery как основа для навигации:

- Выбор узлов (элементы, атрибуты, текст) по иерархическим путям.
- Использование осей (child::, parent::, attribute::).
- Фильтрация узлов через предикаты.
- Поддержка операций с последовательностями узлов.

XQuery использует систему типов XML Schema (XSD). Группы типов:

- Встроенные атомарные типы: xs:string, xs:integer, xs:date, xs:boolean, xs:decimal.
- Типы узлов: element(), attribute(), document-node(), text(), comment().
- Производные типы: определяются в XML Schema.
- Особые типы: xs:untyped (невалидированные данные), xs:anyType.

Типы данных величин узлов и выражений:

- Узлы имеют тип, соответствующий их виду (element, attribute и т.д.).
- Атомарные значения — строки, числа, даты.
- Выражения возвращают последовательности.

Типизация: статическая (на этапе компиляции) или динамическая (время выполнения), зависит от кода.

Проверка типов: через instance of, например:

Извлечение данных:

- data() — атомарное значение узла.
- string() — строковое представление.
- number() — числовое значение.

Последовательность — упорядоченная коллекция узлов и/или атомарных значений. Может быть пустой () или содержать один и более элементов. Не являются вложенными (последовательность последовательностей «выравнивается»).

## Выражения

### Первичные:

- Литералы: "строка", 42, true().
- Переменные: \$var.
- Конструкторы: <tag>{выражение}</tag>.
- Вызовы функций: fn:substring("text", 1, 2).

### Логические:

and, or, not().

### Пути (предикаты, сокращения):

- Пути: /catalog/book/title.
- Предикаты: //book[price > 20][year = 2023].
- Сокращения осей: @ вместо attribute::, // вместо descendant-or-self::.

### Последовательностей:

- Оператор запятая: (1, 2, 3).
- to для диапазонов: 1 to 10.

Арифметические: +, -, \*, div, idiv, mod.

### Сравнения:

- Общие: =, !=, <, > (для последовательностей с existential semantics).
- По значению: eq, ne, lt, gt (для атомарных значений).
- Порядка узлов: <<, >>.

### Функции: особенности вызова и обзор

#### Особенности вызова:

- Префиксное пространство имён (обычно fn:).
- Передача параметров в круглых скобках.
- Поддержка пользовательских функций.

#### Группы и назначение:

- Строковые: fn:concat(), fn:substring(), fn:string-length().
- Числовые: fn:sum(), fn:round(), fn:floor().
- Даты/времени: fn:current-date(), fn:year-from-date().
- Последовательностей: fn:count(), fn:distinct-values(), fn:index-of().
- Узлов: fn:node-name(), fn:root(), fn:path().
- Логические и конструкторы: fn:not(), fn:exists(), fn:empty().
- Агрегатные: fn:avg(), fn:max().
- Пространства имён: fn:namespace-uri().

### Пример запроса XQuery:

xquery

for \$book in //book

where \$book/price > 30

order by \$book/title

return <expensive>{\$book/title/text()}</expensive>

**13. Спецификация XQuery 1.0. Оператор FLWOR: синтаксис, назначение разделов (for, let, where, order by, return), примеры запросов, работа с файлами, подзапросы. Выборка и создание атрибутов. Операторы ветвлений, варианта, кванторы, соединения, пользовательские функции (синтаксис, вызов). Примеры. Переменные, URI и пространства имён.**

Общая характеристика

FLWOR — ключевой оператор XQuery для обработки последовательностей, аналог SQL SELECT-FROM-WHERE. Название образовано от первых букв составляющих его разделов: for, let, where, order by, return.

Базовый синтаксис:

for \$var in выражение

let \$var := выражение

where условие

order by сортировка

return выражение

FOR — итерация по последовательности: создает переменные, связывая их с элементами последовательности. Может быть несколько for.

LET — присваивание переменных: привязывает переменную к целому выражению (не поштучно как for)

WHERE — фильтрация: выбор кортежей, удовлетворяющих условию

ORDER BY — сортировка: упорядочивание результатов

RETURN — формирование результата: конструкция возвращаемого значения

```
for $book in doc("books.xml")/catalog/book
```

```
where $book/price > 30
```

```
order by $book/title
```

```
return <book>
```

```
    {$book/title}
```

```
    <price>{$book/price}</price>
```

```
</book>
```

Загрузка документов:

```
let $doc := doc("library.xml")
```

```
for $book in $doc//book
```

```
return $book/title
```

Подзапросы

Подзапрос в WHERE:

```
for $book in doc("library.xml")//book
```

```
where some $author in $book/author
```

```
    satisfies contains($author, "Толстой")
```

```
return $book/title
```

Выборка атрибутов:

```
for $book in doc("books.xml")//book
```

```
return $book/@isbn
```

Создание атрибутов:

```
for $book in doc("books.xml")//book
return <book isbn="{ $book/@isbn}"
      price="{ $book/price}">
      { $book/title }
</book>
```

Условный оператор (if-then-else):

```
for $book in doc("library.xml")//book
return <book>
      { $book/title }
      <category>{
        if ($book/price > 100) then "premium"
        else if ($book/price > 50) then "standard"
        else "budget"
      }</category>
</book>
```

Оператор typeswitch:

```
typeswitch($node)
case element(book) return <li>{ $node/title/text() }</li>
case element(author) return <strong>{ $node/text() }</strong>
default return $node
```

Операторы варианта (some, every)

Квантор существования (some):

```
for $book in doc("library.xml")//book
where some $author in $book/author
      satisfies contains($author, "Пушкин")
return $book/title
```

Универсальный квантор (every):

```
for $publisher in doc("library.xml")//publisher
where every $book in $publisher/book
      satisfies $book/price < 100
return $publisher/name
```

Соединения (Joins)

Внутреннее соединение:

```
for $book in doc("books.xml")//book,
      $author in doc("authors.xml")//author
where $book/author-id = $author/@id
return <book-with-author>
      { $book/title }
      { $author/name }
</book-with-author>
```

Левое внешнее соединение:

```
for $author in doc("authors.xml")//author
let $books := doc("books.xml")//book[author-id = $author/@id]
return <author-books>
  {$author/name}
  <book-count>{count($books)}</book-count>
  {
    for $book in $books
    return $book/title
  }
</author-books>
```

Пользовательские функции

Объявление функции:

```
declare namespace my = "http://example.com/my-functions";
declare function my:discount-price($price as xs:decimal,
                                   $discount as xs:decimal)
                                   as xs:decimal {
  $price * (1 - $discount div 100)
};
```

```
declare function my:format-price($price as xs:decimal)
                                   as xs:string {
  concat(format-number($price, "#.00"), " руб.")
};
```

Вызов функции:

```
xquery
for $book in doc("books.xml")//book
return <book>
  {$book/title}
  <price>{my:format-price($book/price)}</price>
  <sale-price>{
    my:format-price(my:discount-price($book/price, 15))
  }</sale-price>
</book>
```

Переменные, URI и пространства имен

Объявление переменных:

```
declare variable $default-discount := 10;
declare variable $min-price := 50;
```

```
for $book in doc("books.xml")//book
where $book/price > $min-price
return <book>{$book/title}</book>
```

Пространства имен:

```
declare namespace bib = "http://example.com/bibliography";
declare namespace html = "http://www.w3.org/1999/xhtml";
```

```

<bib:booklist>
{
  for $book in doc("books.xml")//bib:book
  return <html:li>{$book/bib:title}</html:li>
}
</bib:booklist>

```

Работа с URI:

```

declare base-uri "http://example.com/data/";
let $uri := resolve-uri("books.xml")
let $doc := doc($uri)
return count($doc//book)

```

Функция document-uri():

```

for $book in collection("books")//book
let $source := document-uri(root($book))
return <book source="{ $source }">
      { $book/title }
</book>

```

**14. Спецификация XQuery 3.1. Объявления и функции доступа к массивам. Доступ к элементам. Объявления и функции доступа к коллекциям. Изменения в разделах оператора FLWOR (for, window, group by, count). Назначение раздела window: синтаксис объявления, состав и назначение переменных, ключевых слов, формирование групп по значению и позиции элементов. Операторы: try/catch, switch, map.**

XQuery 3.1 — это версия языка, расширяющая XQuery 3.0. Нововведения:

- Поддержка Arrays и Maps как структур данных первого класса.
- Введение операторов для работы с JSON (json-to-xml, xml-to-json).
- Новые операторы: map, switch, улучшенный try/catch.
- Уточнения и расширения в операторе FLWOR.

Массив — это упорядоченная последовательность значений, работающая как единый элемент.

Объявление:

- Квадратные скобки: ["a", "b", 123]
- Функция-конструктор: array { "a", "b", 123 }
- Из последовательности: array { 1 to 5 }

Функции доступа (из пространства имен array:):

- array:size(\$array) — возвращает размер массива.
- array:get(\$array, \$position) — возвращает значение по индексу (начинало с 1).
- array:put(\$array, \$position, \$value) — возвращает новый массив с измененным элементом.
- array:append(\$array, \$value), array:join(\$arrays) и др.

Доступ к элементам (квадратный оператор позиции):

- `$array[$index]` — возвращает последовательность значений по указанным индексам. Например, `[1,2,3][2,1]` вернет 2,1.
- `$array($index)` — синоним `array:get()`, возвращает один элемент. Например, `[1,2,3](2)` вернет 2.

Карта (Map) — это набор пар ключ-значение, где ключ — атомарное значение, а значение — любая последовательность. Объявление:

- Фигурные скобки: `map { "key": "value", 1: "one" }`
- Функция-конструктор: `map:merge(...)`, `map:entry($key, $value)`

Функции доступа (из пространства имен `map:`):

- `map:get($map, $key)` — возвращает значение, ассоциированное с ключом.
- `map:keys($map)` — возвращает все ключи.
- `map:contains($map, $key)` — проверяет наличие ключа.
- `map:put($map, $key, $value)` — возвращает новую карту с добавленной/измененной парой.
- `$map($key)` — возвращает значение по ключу. `map{"x":5}("x")` вернет 5.

Изменения в операторе FLWOR (`for`, `window`, `group by`, `count`):

- `group by`: `group by $key` (переменная привязки или выражение).  
    `for $book in //book`  
    `let $cat := $book/@category`  
    `group by $cat`  
    `return <category name="{ $cat }">{count($book)}</category>`
- `count`: Добавлена как дополнительная переменная в предложении `for`.  
    Позволяет отслеживать номер итерации.  
    `for $item at $count in (1,4,9)`  
    `return <item index="{ $count }">{ $item }</item>`

`window`: Обработка `sliding window` (окна могут пересекаться) и `tumbling window` (окна не могут пересекаться) над последовательностью элементов. Позволяет выполнять вычисления над подмножествами соседних элементов.

```
for tumbling window $w in $seq
  start $s when <условие-начала>
  end $e when <условие-окончания>
  [count <переменная-счетчик>]
  return <выражение>
```

`$w` — переменная, содержащая текущее окно (последовательность элементов).

`$s`, `$e` — переменные, привязанные к начальному и конечному элементу окна.

Состав и назначение переменных и ключевых слов:

`count`: Позволяет задать максимальное количество элементов в окне.

Формирование групп:

- По значению: Условия в start/end используют значения элементов (например,  $\$s/@price > 50$ ).
- По позиции: Условия могут использовать позицию (например,  $\text{count}(\$w) = 3$  в end when для окон фиксированного размера 3).

```
for tumbling window $w in (1 to 10)
start at $s-pos when true()
end at $e-pos when $e-pos - $s-pos = 2
return <window>{$w}</window>
```

try/catch: Обработка динамических (выполняемых) ошибок.

```
try { 1 div 0 } catch * { "Ошибка: " || $err:code }
```

В блоке catch доступны: \$err:code, \$err:description, \$err:value.

switch: Аналог оператора выбора из других языков. Более лаконичен, чем последовательность if-else if.

```
switch ($color)
case "red" return "Стоп"
case "yellow" return "Внимание"
case "green" return "Вперед"
default return "Неизвестно"
```

map (Оператор простого отображения): Применяет функцию или выражение к каждому элементу входной последовательности.

$(1 \text{ to } 5) ! (. * 2) (* \text{ Возвращает } (2,4,6,8,10) *)$

**15. Спецификация SQL/XML. Функции создания XML элементов. Примеры. Функции взаимодействия с XQuery-выражениями, их назначение и особенности реализации в СУБД Oracle и DB2. Примеры. Генерация XML документов в запросах Oracle. Загрузка XML файлов в БД Oracle/DB2.**

SQL/XML — стандарт ISO/IEC, определяющий интеграцию SQL и XML.

-- XMLELEMENT - создание XML элемента

```
SELECT XMLELEMENT("Employee",
    XMLATTRIBUTES(emp_id AS "id"),
    XMLELEMENT("Name", emp_name),
    XMLELEMENT("Department", dept_name)
) AS employee_xml
FROM employees;
```

-- XMLFOREST - создание нескольких элементов

```
SELECT XMLELEMENT("Employee",
    XMLFOREST(
        emp_id AS "id",
        emp_name AS "name",
        salary AS "salary"
    )) FROM employees;
```

Функции взаимодействия с XQuery-выражениями  
В Oracle:

-- XMLQUERY - выполнение XQuery выражения

```
SELECT XMLQUERY('
  for $emp in /Employees/Employee
  where $emp/Salary > 50000
  return $emp/Name'
  PASSING xml_column
  RETURNING CONTENT
) AS high_earners FROM xml_table;
```

-- XMLTABLE - преобразование XML в реляционные данные

```
SELECT x.*
FROM xml_table,
XMLTABLE('/Employees/Employee'
  PASSING xml_column
  COLUMNS
    emp_id NUMBER PATH '@id',
    name VARCHAR2(50) PATH 'Name',
    salary NUMBER PATH 'Salary'
) x;
```

В DB2:

-- XMLEXISTS - проверка существования узлов

```
SELECT * FROM xml_table
WHERE XMLEXISTS('$x/Employee[Salary > 50000]'
  PASSING xml_column AS "x");
```

-- XMLQUERY в DB2

```
SELECT XMLQUERY('
  declare default element namespace "http://example.com";
  $doc/company/employee/name'
  PASSING xml_doc AS "doc"
) FROM xml_documents;
```

-- XMLTABLE в DB2

```
SELECT emp.*
FROM employee_xml,
XMLTABLE('declare namespace ns="http://example.com";
  /ns:Employees/ns:Employee'
  PASSING xml_column
  COLUMNS
    id INTEGER PATH '@id',
    full_name VARCHAR(100) PATH 'ns:Name',
    hire_date DATE PATH 'ns:HireDate'
) AS emp;
```

Генерация XML документов в запросах Oracle

```
DECLARE
    ctx dbms_xmlgen.ctxHandle;
    result CLOB;
BEGIN
    ctx := dbms_xmlgen.newContext(
        'SELECT e.emp_name, e.salary, d.dept_name
        FROM employees e
        JOIN departments d ON e.dept_id = d.dept_id'
    );

    dbms_xmlgen.setRowTag(ctx, 'Employee');
    dbms_xmlgen.setRowSetTag(ctx, 'Employees');

    result := dbms_xmlgen.getXML(ctx);
    dbms_xmlgen.closeContext(ctx);
END;
```

Загрузка XML файлов в БД

-- Создание директории

```
CREATE OR REPLACE DIRECTORY xml_dir AS '/path/to/xml/files';
```

-- Создание таблицы для XML

```
CREATE TABLE xml_documents (
    id NUMBER PRIMARY KEY,
    xml_data XMLTYPE
);
```

-- Загрузка через PL/SQL

```
DECLARE
    xml_file BFILE;
    xml_content CLOB;
BEGIN
    xml_file := BFILENAME('XML_DIR', 'data.xml');
    DBMS_LOB.fileopen(xml_file, DBMS_LOB.file_readonly);
    DBMS_LOB.createtemporary(xml_content, TRUE);
    DBMS_LOB.loadfromfile(xml_content, xml_file, DBMS_LOB.getlength(xml_file));

    INSERT INTO xml_documents VALUES (
        1,
        XMLTYPE(xml_content, nls_charset_id('AL32UTF8'))
    );

    DBMS_LOB.fileclose(xml_file);
    DBMS_LOB.freetemporary(xml_content);
END;
```

В DB2:

-- Создание таблицы

```
CREATE TABLE xml_docs (  
    doc_id INT PRIMARY KEY,  
    doc_data XML  
);
```

-- Загрузка через IMPORT

```
IMPORT FROM 'data.xml' OF XML  
METHOD P (1)  
INSERT INTO xml_docs (doc_id, doc_data)  
VALUES (1, ?);
```

-- Использование LOAD

```
LOAD FROM 'data.xml' OF XML  
MODIFIED BY XMLCHAR  
INSERT INTO xml_docs  
VALUES (1, ?);
```

**16. История создания LINQ. Базовые элементы данных, отложенные и не обложенные операции, нововведения в C#. Функциональное программирование, лямбда-выражения, анонимный тип, переменные var. Варианты синтаксисов и основные разделы операции запросов (from, where, orderby, let, select, group, into, join). Примеры. Операции агрегирования, фильтрации, объединения, преобразования, квантификаторы.**

**17. LINQ to Object. Работа с массивами и коллекциями. Загрузка из файлов. Создание последовательностей и классов коллекций (внешних и внутренних). Примеры. LINQ to XML. Объектная модель, пространства имён. Шаблоны создания элементов. Основные виды запросов. Работа с файлами и столбцами XML. Доступ к значениям и варианты приведения типов. Особенности и достоинства спецификаций.**

**18. LINQ to SQL. Объектное представление (назначение, особенности, методика создания в VS, на основе атрибутов, внешнее сопоставление: XML и DLL). Обработка Null-значений. Подзапросы. Виды и управление загрузками. Запросы в стиле SQL. Достоинства и недостатки LINQ to SQL.**

LINQ to SQL — это технология объектно-реляционного отображения, которая позволяет представлять реляционные данные из SQL Server в виде объектов .NET. Назначение — предоставить возможность работать с данными как с обычными объектами C#/VB.NET, используя синтаксис LINQ.

Особенности:

- Работает только с Microsoft SQL Server
- Автоматическое отображение таблиц на классы, столбцов на свойства
- Поддержка отношений (ассоциаций) между таблицами
- Отслеживание изменений объектов
- Оптимистическая блокировка
- Интеграция с транзакциями

Методика создания в Visual Studio

- Добавление элемента "LINQ to SQL Classes" (\*.dbml файл)
- Drag-and-drop таблиц из Server Explorer на поверхность дизайнера
- Автоматическая генерация классов сущностей и DataContext
- Сохранение файла → автоматическая генерация кода в файле .designer.cs

Создание на основе атрибутов

```
[Table(Name = "Customers")]
public class Customer
{
    [Column(IsPrimaryKey = true)]
    public int CustomerID { get; set; }
    [Column]
    public string Name { get; set; }
    [Column]
    public string City { get; set; }
}
```

Внешнее сопоставление (XML-сопоставление):

```
<Database Name="Northwind">
  <Table Name="Products">
    <Type Name="Product">
      <Column Name="ProductID" Member="ProductID" IsPrimaryKey="true" />
      <Column Name="ProductName" Member="Name" />
    </Type>
  </Table>
</Database>
```

Обработка Null-значений

- Nullable-типы в C# для столбцов, допускающих NULL
- Проверка на null в запросах: `val1 != null`
- Оператор ?? для предоставления значений по умолчанию
- Свойство `HasValue` для проверки nullable-значений

Подзапросы

```
var expensiveProducts = from p in db.Products
    where p.Price > (from avg in db.Products select avg.Price).Average()
    select p;
```

- Отложенная загрузка (Lazy Loading)

Данные загружаются при первом обращении

```
var customer = db.Customers.First(); // Только данные клиента
var orders = customer.Orders; // Загрузка заказов при обращении
```

- Жадная загрузка (Eager Loading)

Загрузка связанных данных одним запросом

```
DataLoadOptions options = new DataLoadOptions();
options.LoadWith<Customer>(c => c.Orders);
db.LoadOptions = options;
```

- Явная загрузка

```
db.LoadProperty(customer, c => c.Orders);
```

Запросы в стиле SQL

```
var customers = db.ExecuteQuery<Customer>(
    "SELECT * FROM Customers WHERE City = {0}", cityName);
```

Достоинства LINQ to SQL:

- Простота использования — минимальная настройка
- Интеграция с IDE — поддержка IntelliSense, отладка
- Типобезопасность — проверка типов на этапе компиляции
- Производительность — кэширование запросов, оптимизация
- Отслеживание изменений — автоматическое управление состоянием объектов
- Декларативный синтаксис — удобство написания запросов

Недостатки:

- Ограниченная поддержка СУБД — только SQL Server
- Производительность на больших объемах уступает чистому ADO.NET
- Отсутствие поддержки наследования
- Устаревание технологии — Microsoft смещает фокус на Entity Framework
- Ограничения в маппинге — не все конструкции БД могут быть отображены