**Курс SQL**

ITVDN - Information Technology Video Developer Network

1. **SQL (Structured Query Language) -** информационно-логический язык, предназначенным для описания, изменения и извлечения данных, хранимых в реляционных базах данных.

***Типы данных***:

1. Целые числа,

2. Числа с фиксированной запятой,

3. Числа с плавающей запятой ,

4. Дата и время,

5. Текстовые данные,

6. Двоичные данные,

7. Пользовательские типы

1. **T-SQL (Transact - Structured Query Language**) – это собственный диалект языка структурированных запрос (SQL), применяемый в СУБД SQL Server.

SQL был расширен такими дополнительными возможностями:

• Управляющие операторы.

• Локальные и глобальные переменные.

• Различные дополнительные функции.

• Поддержка аутентификации Microsoft Windows.

1. **Data Manipulation Language (DML**) – это семейство компьютерных языков, используемых пользователями баз данных для получения, вставки, удаления или изменения данных в базах данных.

Функции DML :

INSERT – вставка данных

SELECT – выборка данных

UPDATE – изменение данных

DELETE – удаление данных

1. **Data Definition Language (DDL)** (язык описания данных) – это словарь, используемый для определения или описания структуры баз данных.

В DDL входят такие операторы как:

Оператор **CREATE** (создать) - используются для определения новых сущностей.

Оператор **ALTER** (изменить) - используются для изменения определений существующих сущностей.

Оператор **DROP** (удалить) - используются для удаления существующих сущностей.

1. **Primary Key** (Первичный Ключ) – предоставляет ссылку для связи с другими таблицами, а так же задает ограничение уникальности для столбца, на котором задается.
2. **Нормализация таблиц** – это формальный аппарат ограничений на формирование таблиц, который позволяет устранить дублирование данных, обеспечивает непротиворечивость хранимых в базе данных, уменьшает трудозатраты на ведение базы данных (ввод и корректировку данных).

Для нормализации таблиц БД используют ограничения:

• Первой нормальной формы.

• Второй нормальной формы.

• Третей нормальной формы.

**Первая нормальная форма** (1NF) – отсутствие повторяющихся данных, любое поле любой записи хранит только одно значение.

**Вторая нормальная форма** (2NF) – требует предварительного приведения исходной таблицы к первой НФ, а так же каждый не ключевой столбец таблицы находящейся в 1НФ, должен зависеть от всего ключа.

**Третья нормальная форма** (3NF) – требует предварительного приведения исходной таблицы ко второй НФ, а так же ни в одном не ключевом столбце не может быть зависимости от другого не ключевого столбца. Так же не допускается наличие в таблице производных данных.

1. **Денормализация** – процесс понижения нормальной формы. Осуществляется если приведенная высшая форма приводит к ухудшению практического использования.
2. **JOIN** (объединение таблиц)

**INNER JOIN** (Внутреннее объединение) – берется общее из двух таблиц - объединение, при котором в запросе все записи из таблицы на левой и правой стороне операции INNER JOIN добавляются в результирующий набор записей, при соответствии условию значений в связанных полях.

**LEFT JOIN** (Объединение по левой таблице) – берется левая таблица + общее из двух

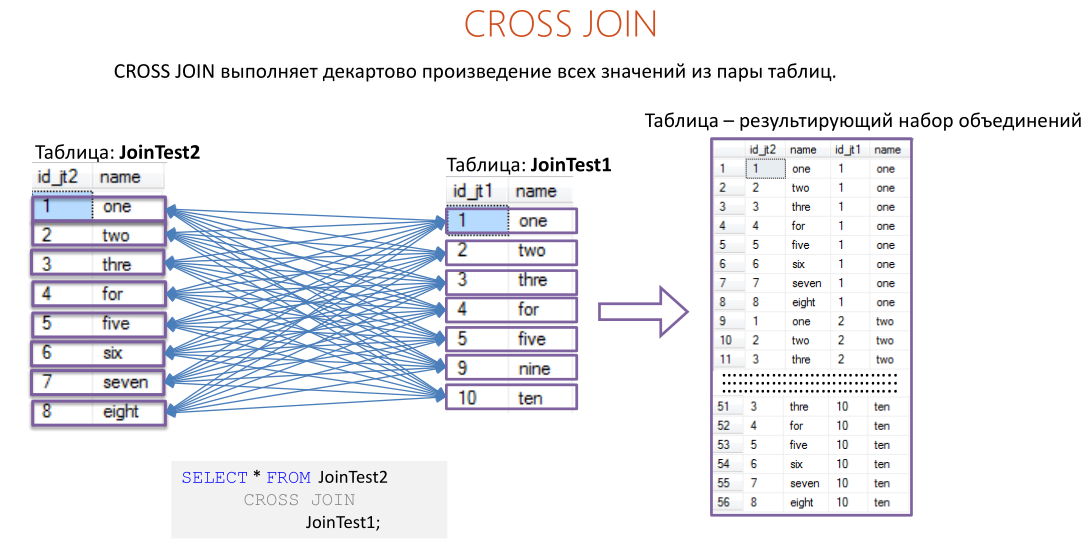
**LEFT OUTER JOIN** (Левое внешнее объединение) - внешнее объединение, при котором в запросе все записи из таблицы на левой стороне операции LEFT JOIN в инструкции SQL добавляются в результирующий набор записей, даже если в таблице на правой стороне отсутствуют совпадающие значения в связанных полях.

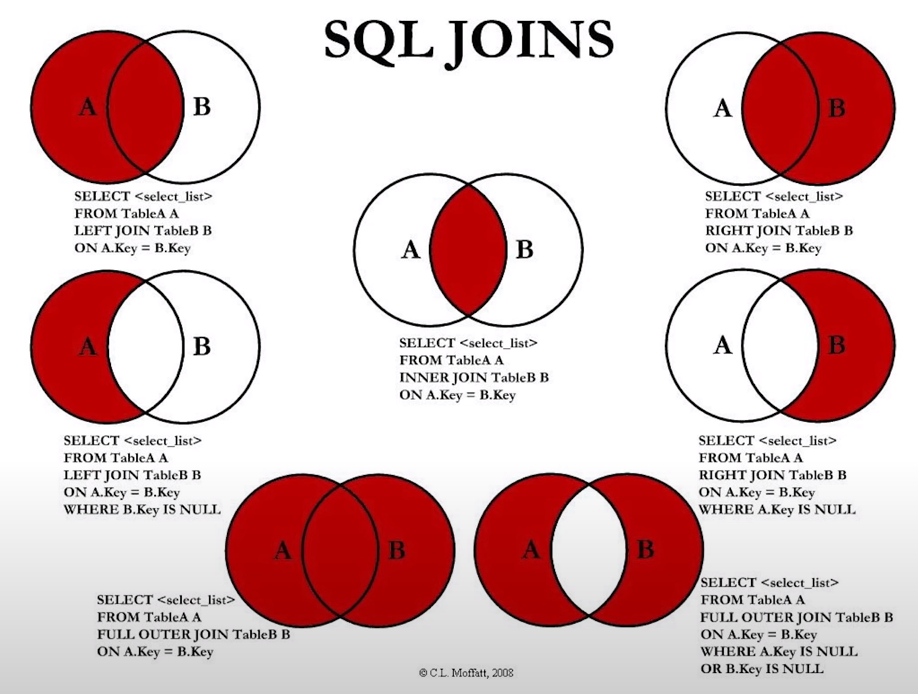
**RIGHT JOIN** (Объединение по правой таблице) – берется правая таблица + общее из двух

**RIGHT OUTER JOIN** (Правое внешнее объединение) - внешнее объединение, при котором в запросе все записи из таблицы на правой стороне операции RIGHT JOIN в инструкции SQL добавляются в результирующий набор записей, даже если в таблице на левой стороне отсутствуют совпадающие значения в связанных полях.

**FULL OUTER JOIN** (Полное внешнее объединение) – берутся обе таблицы полностью

**CROSS JOIN** (перекрёстное объединение) - декартовое произведение таблиц

В CROSS JOIN не используется конструкция ON



**UNION** объединяет результаты двух запросов SELECT в единую результирующую таблицу.

Если результаты обоих запросов содержат строки с совпадающими значениями ячеек, то, операция UNION помещает в результирующую таблицу только одну такую строку.

Если в результате одного из запросов имеются строки с уникальными значениями, не совпадающими ни с одной из строк результата другого запроса, то эта строка так-же помещается в результирующую таблицу.

Операция UNION требует использования таких запросов, каждый из которых возвращает выборку в табличном представлении, при этом, типы и количество столбцов должны совпадать.

SELECT \* FROM JoinTest1

UNION

SELECT \* FROM JoinTest2

**UNION ALL** объединяет результаты двух запросов SELECT в единую результирующую таблицу.

Если результаты обоих запросов содержат строки с совпадающими значениями ячеек, то, операция UNION ALL помещает в результирующую таблицу все дублирующиеся строки.

Если в результате одного из запросов имеются строки с уникальными значениями, не совпадающими ни с одной из строк результата другого запроса, то эта строка так-же помещается в результирующую таблицу.

Операция UNION ALL требует использования таких запросов, каждый из которых возвращает выборку в табличном представлении, при этом, типы и количество столбцов должны совпадать.

**EXCEPT** исключает результаты правого запроса.

Если результат левого запроса операции EXCEPT содержит уникальные строки, не совпадающие ни с одной из строк правого запроса, то, только такие строки помещается в результирующую таблицу.

Уникальные строки правого запроса операции EXCEPT, никогда не входят в результирующую таблицу.

Если результаты обоих запросов содержат совпадающие строки, то, операция EXCEPT игнорирует их.

Операция EXCEPT требует использования таких запросов, каждый из которых возвращает выборку в табличном представлении, при этом, типы и количество столбцов должны совпадать.

SELECT \* FROM JoinTest1

EXCEPT

SELECT \* FROM JoinTest2

**INTERSECT** объединяет результаты двух запросов SELECT в единую результирующую таблицу.

Если результаты обоих запросов содержат строки с совпадающими значениями ячеек, то, операция INTERSECT помещает в результирующую таблицу только одну такую строку.

Если в результате одного из запросов имеются уникальные строки, не совпадающие ни с одной из строк результата другого запроса, то такие строки игнорируются операцией INTERSECT.

SELECT \* FROM JoinTest1

INTERSECT

SELECT \* FROM JoinTest2

1. **Subqueries** (Подзапросы или вложенные запросы) – это запросы, используемые в инструкциях SELECT, INSERT, UPDATE или DELETE

SELECT ProductID, Name

FROM Product

WHERE Color NOT IN

(SELECT Color

FROM Product

WHERE ProductID = 5)

1. **Related** **Subqueries** (связанные подзапросы) - Подзапрос является связанным, если в нем (в предложениях WHERE, HAVING) указан столбец таблицы внешнего запроса.

SELECT ord1.OrderDate

FROM Order AS ord1

WHERE ord1.OrderDate =

(SELECT MIN(OrderDate)

FROM Order AS ord2

WHERE ord2.CustomerID = ord1.CustomerID)

Правила написания подзапросов:

• Вложенные запросы могут быть указаны в инструкциях: SELECT, INSERT, UPDATE или DELETE.

• Каждый подзапрос может содержать один или более подзапросов.

• Предложение WHERE внешнего запроса должно быть совместимо для соединения со столбцом в списке выбора вложенного запроса.

1. **Страницы**  – основная единица хранения информации в SQL Server.

Основные типы страниц баз данных SQL Server:

1. **Страницы данных** – содержат большую часть данных, которые выводятся в

таблицы. На странице данных находятся строки данных со всеми данными,

кроме данных типа text, ntext, image, nvarchar(max), varchar(max) и varbinary(max).

1. **Страницы индексов** – используются для ускорения доступа к данным. Содержат

списки значений одного или нескольких полей таблицы и их ассоциации с

номерами записей самой таблицы.

1. **Экстент** – основная единица распределения памяти в Базе Данных
2. **Куча** – таблица, которая не имеет кластеризованного индекса. Строки данных хранятся без определенного порядка и какой-либо порядок в последовательности страниц данных отсутствует.
3. **Индекс** – объект базы данных, создаваемый с целью повышения производительности поиска данных.

Ускорение работы с использованием индексов достигается в первую очередь за счёт того, что индекс имеет структуру, оптимизированную под поиск – например, сбалансированного дерева.

**Типы индексов:**

1. **Кластеризированный индекс** – упорядоченный индекс, смешанный с данными. На листовом уровне находятся все действительные данные таблицы
2. **Некластеризованный индекс** – неупорядоченный индекс, на листовом уровне которого находится идентификатор строки (RID) указывающий на место хранения остальных данных.

• заданный на неупорядоченной таблице

• заданный на кластеризированной таблице

1. **Хранимые процедуры** (Stored Procedure)

Хранимые процедуры в Microsoft SQL Server аналогичны процедурам в других языках программирования:

• они обрабатывают входные параметры и возвращают вызывающей процедуре или пакету значения в виде выходных параметров;

• они содержат программные инструкции, которые выполняют операции в базе данных, в том числе вызывающие другие процедуры;

• они возвращают значение состояния вызывающей процедуре или пакету, таким образом передавая сведения об успешном или неуспешном завершении (и причины последнего).

**Создание хранимой процедуры** (Create Procedure)

CREATE PROC spEmployee

-- Создание хранимой процедуры.

AS

SELECT \* FROM Employee;

GO

EXEC spEmployee; --Вызов хранимой процедуры.

GO

1. **Пользовательские функции** (User-Defined Function) - представляет собой подпрограмму, которая принимает параметры, выполняет действия, такие как сложные вычисления, а затем возвращает результат этих действий в виде значения. Возвращаемое значение может быть скалярным значением или таблицей.

CREATE FUNCTION Hello() -- создать функцию

RETURNS varchar(30)

-- объявляем тип возвращаемого значения

AS

BEGIN -- начало тела функции

DECLARE @MyVar varchar(20) = 'Hello World!';

RETURN @MyVar; --возвращаемое значение функции

END; -- конец тела функции

GO

PRINT dbo.Hello();

1. **Транзакция** - выполнение последовательности команд (SQL-конструкций) в базе данных, которая либо фиксируется при успешной реализации каждой команды, либо отменяется при неудачном выполнении хотя бы одной команды.

BEGIN TRANSACTION;

DECLARE @Id int;

INSERT MyUserName VALUES ('TestName0','TestLName0');

SET @Id = @@IDENTITY;

INSERT MyUserTell VALUES (@Id,'(097)2224455');

COMMIT TRANSACTION;

**Откат транзакции** – это действие, обеспечивающее аннулирование всех изменений данных, которые были сделаны в теле текущей незавершенной транзакции.

BEGIN TRANSACTION;

DECLARE @Id int;

INSERT MyUserName VALUES ('TestName0','TestLName0');

SET @Id = @@IDENTITY;

INSERT MyUserTell VALUES (@Id,'(097)2224455');

ROLLBACK TRANSACTION;

При выполнении транзакций несколькими пользователями одной базы данных могут возникать следующие проблемы:

**• Lost update**

**• Dirty reads**

**• Non-repeatable reads**

**• Phantom reads**

**17.1 Lost update** – потерянное обновление. При одновременном изменении одного блока данных разными транзакциями одно из изменений теряется.

В обеих транзакциях изменяется значение поля f2, при этом одно из изменений теряется. Так что, f2 будет увеличено не на 45, а только на 20 или 25.

Это происходит потому, что:

1. Первая транзакция прочитала текущее состояние поля.

2. Вторая транзакция сделала свои изменения, основываясь на своих

сохраненных в память данных.

3. Первая делает обновление поля, используя свои «старые» данные.

**17.2 Dirty reads –** «грязное» чтение. Чтение данных, добавленных или изменённых транзакцией, которая впоследствии не подтвердится (откатится).

Первый пользователь начинает транзакцию, изменяющую данные. В это время другой пользователь (или создаваемая им транзакция) извлекает частично измененные данные, которые не являются корректными

**17.3** **Non-repeatable reads** – неповторяемое чтение. Ситуация, когда при повторном чтении в рамках одной транзакции ранее прочитанные данные оказываются изменёнными.

Первый пользователь начинает транзакцию, изменяющую данные. В это время другой пользователь начинает и завершает другую транзакцию. Первый пользователь при повторном чтении данных (например, если в его транзакцию входит несколько инструкций SELECT) получает другой набор записей.

**17.4 Phantom reads** – чтение фантомов. Ситуация, когда при повторном чтении в рамках одной транзакции одна и та же выборка дает разные множества строк.

Первый пользователь начинает транзакцию, выбирающую данные из таблицы. В это время другой пользователь начинает и завершает транзакцию, вставляющую или удаляющую записи. Первый пользователь получит другой набор данных, содержащий фантомы – удаленные или измененные строки.

Для решения этих проблем разработаны четыре уровня изоляции транзакции в SQL:

• **Read uncommitted.** Транзакция может считывать данные, с которыми работают другие транзакции. Применение этого уровня изоляции может привести ко всем перечисленным проблемам.

• **Read committed**. Транзакция не может считывать данные, с которыми работают другие транзакции. Применение этого уровня изоляции исключает проблему «грязного» чтения.

• **Repeatable read.** Транзакция не может считывать данные, с которыми работают другие транзакции. Другие транзакции также не могут считывать данные, с которыми работает эта транзакция. Применение этого уровня изоляции исключает все проблемы, кроме чтения фантомов.

• **Serializable**. Транзакция полностью изолирована от других транзакций. Применение этого уровня изоляции полностью исключает все проблемы.

Поведение при различных уровнях изолированности:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Уровень изолированности | Lost update  (потерянное обновление) | Dirty reads  («грязное» чтение) | Non-repeatable reads  (неповторяемое чтение) | Phantom reads  (чтение фантомов) |
| READ UNCOMMITTED |  | Разрешено | Разрешено | Разрешено |
| READ COMMITTED |  |  | Разрешено | Разрешено |
| REPEATABLE READ |  |  |  | Разрешено |
| SERIALIZABLE |  |  |  |  |

1. **Триггер** – это обработчик который можно выполнить во время выполнения операций INSERT, UPDATE, DELETE

Вместе с созданием триггера постоянно создаются две служебные таблицы: inserted и deleted