Other functions

```
-- Sample Products table
CREATE TABLE Products (
  ProductID INT,
  ProductName VARCHAR(50),
  Price DECIMAL(10, 2)
);

-- Insert data into the table
INSERT INTO Products (ProductID, ProductName, Price)
VALUES
  (1, 'Widget', 10.00),
  (2, 'Gadget', NULL),
  (3, 'Doodad', 25.50);

-- Example using IIF to categorize products
SELECT
  ProductID,
  ProductName,
  Price,
  IIF(Price IS NULL, 'Price not available', 'Price available') AS PriceCategory
FROM Products;
```

Convert Data

```
SELECT * FROM Students
WHERE StudentID='1'


SELECT CAST(12.50 AS INT)

SELECT CONVERT(VARCHAR,GETDATE(),101)

SELECT GETDATE()

SELECT COALESCE(NULL,NULL,'Deepthi',NULL,'Narayan')

SELECT TRY_CAST('ABC' AS INT)
```

```
CREATE TABLE Products(
    ProductID INT,
    ProductName varchar(50),
    Price varchar(50),
    ProductDescription varchar(100)
)

INSERT INTO Products VALUES(1,'Laptop','8000','Awesome'),
(2,'Mobile','10000',NULL)

SELECT ProductID, ProductName,PRICE,
COALESCE(ProductDescription,'No Description') AS ProductDesciption
FROM Products
```

Joins and Unions

Sure, here's the difference between joins and unions in simple language:

Joins: Think of joins like combining puzzle pieces. When you have two tables, you use a join to find matching pieces between them. It's like connecting related information. For example, if you have a list of students and a list of classes, a join helps you figure out which student is in which class.

Unions: Unions are like stacking things on top of each other. When you have two similar lists, you use a union to put them together, making one big list. For instance, if you have a list of books in one place and another list of books in another place, a union lets you create a single list with all the books from both places combined.

```
CREATE TABLE BOOK1(
    BookID INT,
    Title varchar(50)
)

CREATE TABLE BOOK2(
    BookID INT,
    Title varchar(50)
)

INSERT INTO BOOK1 VALUES(1,'The Great Gatsby'),(2, 'To Kill a MockingBird')
INSERT INTO BOOK2 VALUES(3,'1984'),(2, 'Animal Farm')

SELECT * FROM BOOK1
SELECT * FROM BOOK2

SELECT * FROM BOOK1
UNION
SELECT * FROM BOOK2
```

```
CREATE TABLE Customers(
    CustomerID INT,
    City varchar(50),
    Country varchar(50)
```

```
)

CREATE TABLE Suppliers(
    CustomerID INT,
    City varchar(50),
    Country varchar(50)
)

INSERT INTO Customers values(1,'Berlin','Germany'),(2,'Munich','Germany'),
(3,'Hamburg','Germany'),(4,'Paris','France'),(5,'London','Uk')
INSERT INTO Suppliers values(101,'Frankfurt','Germany'),(102,'Berlin','Germany'),
(103,'Munich','Germany'),(104,'Milan','Italy'),(5,'London','Uk')

--SELECT City,Country FROM Customers
--UNION
--SELECT City,Country FROM Suppliers

SELECT City,Country FROM Customers
WHERE Country ='GERMANY'
UNION
SELECT City,Country FROM Suppliers
WHERE Country ='GERMANY'
```

Aggregate functions

```
CREATE TABLE Restaurants(
RestaurantID INT PRIMARY KEY,
RestaurantName VARCHAR(100) NOT NULL,
Cuisine VARCHAR(50) NOT NULL,
Location VARCHAR(100) NOT NULL,
Rating FLOAT
)

INSERT INTO Restaurants VALUES (1, 'Pizza Hut', 'Margoretta', 'Bangalore', 5)
INSERT INTO Restaurants VALUES (2, 'Dominos', 'Margoretta', 'Bangalore', 4)
INSERT INTO Restaurants VALUES (3, 'Delhi Belly', 'Margoretta', 'Bangalore', 3)
-- Inserting data into the Restaurants table
INSERT INTO Restaurants (RestaurantID, RestaurantName, Cuisine, Location, Rating)
VALUES
  (4, 'Tasty Bites', 'Italian', 'Downtown', 4.5),
  (5, 'Spice Delight', 'Indian', 'Midtown', 4.0),
  (6, 'Sushi Haven', 'Japanese', 'Uptown', 4.2);


--SELECT Cuisine, AVG(Rating)
--FROM Restaurants

SELECT Cuisine, AVG(Rating)
FROM Restaurants
GROUP BY Cuisine
HAVING AVG(Rating)>=4.2

SELECT Cuisine, MIN(Rating)
FROM Restaurants
```

```
GROUP BY Cuisine
```

```
CREATE TABLE Restaurants(
RestaurantID INT PRIMARY KEY,
RestaurantName VARCHAR(100) NOT NULL,
Cuisine VARCHAR(50) NOT NULL,
Location VARCHAR(100) NOT NULL,
Rating FLOAT
)

DROP TABLE Restaurants


INSERT INTO Restaurants (RestaurantID, RestaurantName, Cuisine, Location, Rating)
VALUES
  (1, 'Tasty Bites', 'Italian', 'Downtown', 4.5),
  (2, 'Spice Delight', 'Indian', 'Midtown', 4.0),
  (3, 'Sushi Haven', 'Japanese', 'Uptown', 4.2),
  (4, 'Burger Joint', 'American', 'Downtown', 3.8),
  (5, 'Curry House', 'Indian', 'Suburb', 4.1),
  (6, 'Pizza Palace', 'Italian', 'Uptown', 4.3);


--SELECT Cuisine, AVG(Rating)
--FROM Restaurants

SELECT Cuisine, AVG(Rating)
FROM Restaurants
GROUP BY Cuisine
HAVING AVG(Rating)>=4.2

SELECT Cuisine, MIN(Rating)
FROM Restaurants
GROUP BY Cuisine


SELECT Cuisine, Location,AVG(Rating )
FROM Restaurants
GROUP BY ROLLUP(Cuisine, Location)

SELECT *
FROM Restaurants
PIVOT (
    AVG(Rating)  -- Aggregation function
    FOR Cuisine IN ([Italian], [Japanese], [American], [Indian])  -- List of
cuisine values
) AS PivotTable;
```

```
SELECT *
FROM EmployeeSalaries
PIVOT (
    AVG(Salary) AS AvgSalary,
    MAX(Salary) AS MaxSalary,
    MIN(Salary) AS MinSalary
    FOR Department IN ([HR], [IT], [Finance], [Marketing])  -- List of department
values
) AS PivotTable;
```

```
SELECT *
FROM StudentGrades
PIVOT (
    AVG(Grade)  -- Aggregation function (average)
    FOR Subject IN ([Math], [Science], [History], [English])  -- List of subject
values
) AS PivotTable;
```

Views

```
CREATE VIEW HighRatedItalianRestaurant AS
SELECT RestaurantName,Location,Rating
FROM Restaurants
WHERE Cuisine='Italian' AND Rating >=4.5

SELECT * FROM HighRatedItalianRestaurant
```

Certainly, here's an example with sample tables and the expected result for the view:

Students Table:

StudentID	FirstName	LastName
1	John	Smith
2	Jane	Doe
3	Michael	Johnson

Courses Table:

CourseID	CourseName	StudentID
101	Math	1
102	Science	1
103	History	2
104	English	3

CourseID	CourseName	StudentID
105	Math	3

Creating the View:

```
CREATE VIEW StudentCourseEnrollments AS
SELECT
    s.StudentID,
    s.FirstName,
    s.LastName,
    c.CourseName
FROM
    Students s
JOIN
    Courses c ON s.StudentID = c.StudentID;
```

Using the View:

```
SELECT * FROM StudentCourseEnrollments;
```

Expected Result:

StudentID	FirstName	LastName	CourseName
1	John	Smith	Math
1	John	Smith	Science
2	Jane	Doe	History
3	Michael	Johnson	English
3	Michael	Johnson	Math

In the "StudentCourseEnrollments" view, each student's full name is combined with the courses they are enrolled in. The view uses a JOIN between the "Students" and "Courses" tables to achieve this. When you query the view, you get a consolidated result showing student names along with the courses they are enrolled in.

Certainly, here's another example involving sample tables and a view that uses a different logic.

Orders Table:

OrderID	CustomerID	OrderDate	TotalAmount
1	101	2023-07-01	150.00
2	102	2023-07-02	75.00
3	101	2023-07-02	200.00
4	103	2023-07-03	120.00

OrderID	CustomerID	OrderDate	TotalAmount
5	102	2023-07-04	50.00

Customers Table:

CustomerID	FirstName	LastName
101	John	Smith
102	Jane	Doe
103	Michael	Johnson

Creating the View:

Here, we will create a view that shows each customer's total spending based on their orders:

```
CREATE VIEW CustomerTotalSpending AS
SELECT
    c.CustomerID,
    c.FirstName,
    c.LastName,
    SUM(o.TotalAmount) AS TotalSpending
FROM
    Customers c
JOIN
    Orders o ON c.CustomerID = o.CustomerID
GROUP BY
    c.CustomerID, c.FirstName, c.LastName;
```

Using the View:

```
SELECT * FROM CustomerTotalSpending;
```

Expected Result:

CustomerID	FirstName	LastName	TotalSpending
101	John	Smith	350.00
102	Jane	Doe	125.00
103	Michael	Johnson	120.00

In this example, the "CustomerTotalSpending" view combines customer details from the "Customers" table with their total spending calculated based on orders from the "Orders" table. The result is a view showing each customer's total spending.

Case statement

Certainly! A CASE statement in SQL is used for conditional logic within queries. Let's consider an example using the "Employees" table to demonstrate how a CASE statement works.

Employees Table:

EmployeeID	FirstName	LastName	Salary
101	John	Smith	55000
102	Jane	Doe	62000
103	Michael	Johnson	48000
104	Emily	Williams	70000
105	Robert	Brown	60000

Suppose you want to categorize employees into salary brackets based on their salary. Here's how you might use a CASE statement to achieve this:

```
SELECT
    EmployeeID,
    FirstName,
    LastName,
    Salary,
    CASE
        WHEN Salary >= 65000 THEN 'High'
        WHEN Salary >= 55000 THEN 'Medium'
        ELSE 'Low'
    END AS SalaryCategory
FROM Employees;
```

Expected Result:

EmployeeID	FirstName	LastName	Salary	SalaryCategory
101	John	Smith	55000	Medium
102	Jane	Doe	62000	High
103	Michael	Johnson	48000	Low
104	Emily	Williams	70000	High
105	Robert	Brown	60000	Medium

In this example, the CASE statement evaluates the "Salary" column and assigns a category based on the salary range. Depending on the salary, the result will have a "SalaryCategory" column indicating whether the salary is "High," "Medium," or "Low."