

SQL Server For Backend Development

UNIT-5

INTRODUCTION TO DATABASE

INTRODUCTION TO DATABASE

- A database is information that is set up for easy access, management and updating.
- It is an organized collection of data,
- generally stored and accessed electronically from a computer system.
- It supports the storage and manipulation of data.
- In other words, databases are used by an organization as a method of storing, managing and retrieving information.
- To manage these databases, Database Management Systems (DBMS) are used.

Flow of Database

INTRODUCTION TO DATABASE



Advantages of Database Management System

- The data is stored in a neater way and hence, more data can be stored.
- A DBMS is a highly secure platform so confidential and high-risk data can also be stored and accessed, securely.
- DBMS makes handling of data very simple.
- Data inconsistency is greatly reduced by a well-designed DBMS.
- Data can be accessed quickly.

INTRODUCTION TO SQL SERVER

Microsoft SQL Server is a relational database management system ([RDBMS](#)) that supports a wide variety of [transaction](#) processing, business intelligence ([BI](#)) and data analytics applications in corporate IT environments.

Like other RDBMS software, Microsoft SQL Server is built on top of Structured Query Language ([SQL](#)), a standardized programming language that database administrators ([DBAs](#)) and other IT professionals use to manage databases and query the data they contain. SQL Server is tied to Transact-SQL ([T-SQL](#)), Microsoft's proprietary query language that enables applications and tools to communicate and also connect to a SQL Server instance or database.

Usage of sql server

USAGE OF SQL SERVER

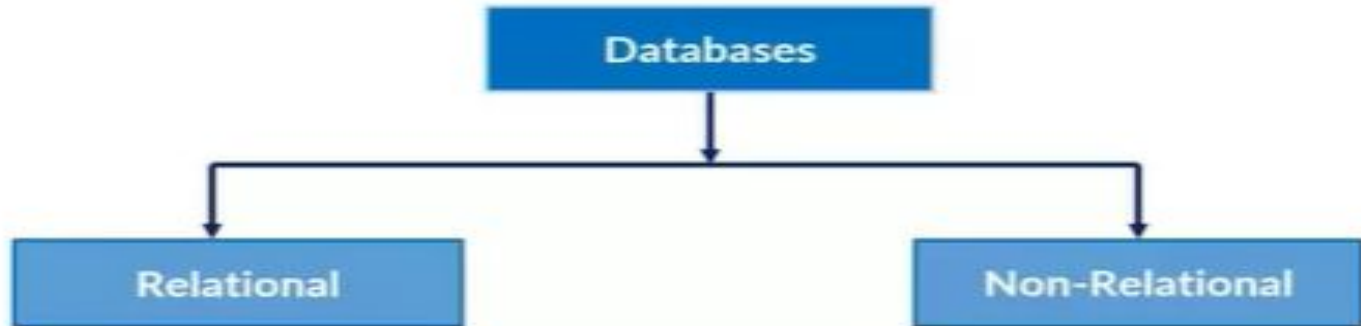
The following are the key usage of MS SQL Server:

- Its main purpose is to build and maintain databases.
- It is used to analyze the data using SQL Server Analysis Services (SSAS).
- It is used to generate reports using SQL Server Reporting Services (SSRS).
- It is used to perform ETL operations using SQL Server Integration Services (SSIS).

Types of Databases

TYPES OF DATABASE

There are two common types of databases:



RDBMS

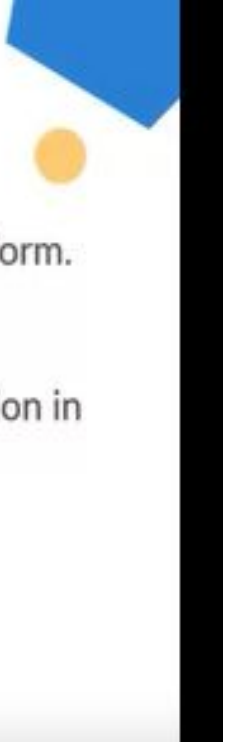
RELATIONAL DATABASE

- Relational database stores information in tables containing specific pieces and types of data.
- **For example**, a shop could store details of their customers' names and addresses in one table and details of their orders in another.
- This form of data storage is often called structured data.
- Relational databases use Structured Query Language (SQL).
- In relational database, the database usually contains tables consisting of columns and rows.
- Relational databases work best when the data they contain doesn't change very often, and when accuracy is crucial.

Non-Relational Database

NON -RELATIONAL DATABASE

- Non-relational databases are often called NoSQL databases.
- They are different from relational databases in that they store their data in a non-tabular form.
- Instead, non-relational databases might be based on data structures like documents.
- A document can be highly detailed while containing a range of different types of information in different formats.
- This ability to digest and organize various types of information side by side makes non-relational databases much more flexible than relational databases.



SQL SERVER COMMANDS

SQL Server Commands

- There are five types of SQL commands: DDL, DML, DCL, TCL, and DQL.

DDL – Data Definition Language

DML – Data Manipulation Language

DCL – Data Control Language

TCL – Transactional Control Language

DQL – Data Query Language

DDL Commands

DDL COMMANDS

- DDL is abbreviation of Data Definition Language.
- It is used to create and modify the structure of database objects in database.
- **CREATE** – Creates objects in the database.
- **ALTER** – Alters objects of the database.
- **DROP** – Deletes objects of the database.
- **TRUNCATE** – Deletes all records and resets table to initial stage.

DML COMMANDS


- DML commands are used to modify the database.
- It is responsible for all form of changes in the database.
- The command of DML is not auto-committed that means it can't permanently save all the changes in the database.
- They can be rollback.
- Here are some commands that come under DML:
- **INSERT** - used to insert data into the row of a table.
- **UPDATE** - used to update or modify the value of a column in the table.
- **DELETE** - used to remove one or more row from a table.

DCL COMMANDS

- DCL commands are used to grant and take back authority from any database user.
- Here are some commands that come under DCL:
- Grant - used to give user access privileges to a database.
- Revoke - used to take back permissions from the user.

TCL COMMANDS



- TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only. 
- These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.
- Here are some commands that come under TCL:
- COMMIT - used to save all the transactions to the database.
- ROLLBACK - used to undo transactions that have not already been saved to database.
- SAVEPOINT - used to roll transaction back to a certain point without rolling back the entire transaction.

CONSTRAINTS

SQL SERVER CONSTRAINTS

- In a database table, we can add rules to a column known as **constraints**.
- These rules control the data that can be stored in a column.
- Otherwise, insert operation will be terminated if inserted data violates the defined constraint.
- They are responsible for ensuring column's data accuracy, integrity, and reliability inside table.
- For example, if a column has NOT NULL constraint, it means the column cannot store NULL values.



SQL SERVER CONSTRAINTS



SQL Server categorizes the constraints into two types:

- **Table level constraints:**
 - These constraints apply to entire table that limit the types of data that are entered into table.
 - Its definitions are specified after creating the table using the ALTER statement.
- **Column level constraints:**
 - These constraints apply to the single or multiple columns to limit the types of data that can be entered into the column.
 - Its definition is specified while creating the tables.

SQL SERVER CONSTRAINTS

The constraints used in SQL Server are:

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT

PRIMARY KEY CONSTRAINT



- This constraint consists of one or more columns with values and identifies each record uniquely.
- We cannot enter the null, empty, or duplicate values in the primary key constraints columns.
- Each table can contain only one primary key column; however, it can have duplicate columns.
- It always contains unique values in a column.
- We mainly used this constraint to enforce the entity integrity of the table.
- Syntax:

```
CREATE TABLE table_name (  
    column1 data_type,  
    ...,  
    [CONSTRAINT constraint_name] PRIMARY KEY (column1)  
);
```

PRIMARY KEY CONSTRAINT

```
CREATE TABLE Students (  
    StudentID INT PRIMARY KEY,  
    Name VARCHAR(50),  
    Age INT  
);
```

FOREIGN KEY CONSTRAINT



- FOREIGN KEY constraint is different from PRIMARY KEY constraint as each table can only have one PRIMARY KEY.
- Each table can have several FOREIGN KEY constraints by referring to multiple parent tables.
- Also, we cannot insert NULL values in the primary key column,
- Syntax:

```
CREATE TABLE table_name (  
    column1 data_type,  
    column2 data_type,  
    ....  
    FOREIGN KEY (column_name)  
    REFERENCES referenced_table_name (referenced_column_name)  
);
```

FOREIGN KEY CONSTRAINT

Enforces referential integrity by ensuring that the values in a column or columns of one table exist in another table.

```
CREATE TABLE Orders (
```

```
    OrderID INT PRIMARY KEY,
```

```
    ProductID INT,
```

```
    Quantity INT,
```

```
    CONSTRAINT FK_ProductID FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
```

```
);
```

CHECK CONSTRAINT

- This constraint is used to limit the range of values in a column.
- It ensures that all the inserted values in a column must follow the specific rule.
- It controls value in a particular column and assures no corrupted information is entered in a column.
- We can specify more than one check constraint for a specific column.
- When a NULL value is included in the condition, this constraint will yield an UNKNOWN value.
-

```
CREATE TABLE table_name (  
    column_name data_type CHECK(condition)  
);
```

CHECK CONSTRAINT

- Enforces that values in a column meet a specific condition.
- Example:

```
CREATE TABLE Products (  
    ProductID INT PRIMARY KEY,  
    ProductName VARCHAR(50),  
    Price DECIMAL(10, 2),  
    CONSTRAINT CHK_Price CHECK (Price >= 0)  
);
```

DEFAULT CONSTRAINT



- This constraint is used to insert the default value in the column when the user does not specify any value for that column.
- It helps to maintain domain integrity when no value is provided into the specified default constraint column.
- It assures that column must contain a value including constant, system-defined value, or NULL.
- It can also be created either during the creation of the table or after creating the table.
- Syntax:

```
CREATE TABLE table_name (  
    column_name data_type DEFAULT default_value  
);
```


DEFAULT CONSTRAINT

- Specifies a default value for a column when no value is explicitly provided during an INSERT operation.
- Example:

```
CREATE TABLE Employees (  
    EmployeeID INT PRIMARY KEY,  
    EmployeeName VARCHAR(50),  
    Department VARCHAR(50) DEFAULT 'IT'  
);
```

DEFAULT CONSTRAINT EXAMPLE

```
USE palvidb1;
```

```
CREATE TABLE students (
```

```
    student_id INT PRIMARY KEY,
```

```
    student_name VARCHAR(100),
```

```
    grade CHAR(1) DEFAULT 'A'
```

```
);
```

```
INSERT INTO students (student_id, student_name) VALUES (1, 'John');
```

```
SELECT * FROM STUDENTS;
```

```
INSERT INTO students (student_id, student_name, grade) VALUES (2, 'Jane', 'B');
```

```
SELECT * FROM STUDENTS;
```

NOT NULL

Not Null Constraint:

- Ensures that a column does not accept NULL values.
- Example:

```
CREATE TABLE Customers (  
    CustomerID INT PRIMARY KEY,  
    CustomerName VARCHAR(50) NOT NULL,  
    Email VARCHAR(50)  
);
```

Example of Constraints

SQLQuery1.sql - D:\...-600KGAO\HP (55))*

```
Use palvidb1;
create table T1(
    Dept_id int primary key,
    Dept_name varchar not null,
    Dept_location varchar(500) check(Dept_location IN('Punjab','Goa','UP')),
    Dept_AddedDate Datetime Default GetDate());

create table t2(
    EmpId int, Emp_name varchar(50), Emp_salary float, Emp_Email varchar(200)
);
```

Use palvidb1;

create table T1(

Dept_id int primary key,

Dept_name varchar not null,

Dept_location varchar(500) check(Dept_location IN('Punjab','Goa','UP')),

Dept_AddedDate Datetime Default GetDate());

create table t2(

EmpId int, Emp_name varchar(50), Emp_salary float, Emp_Email varchar(200)

);

Tasks Given To the Students

You're tasked with designing a database for a library management system. The system needs to track information about books, authors, and borrowers. Each book has a unique ISBN (International Standard Book Number) assigned to it, and each borrower has a unique library card number. Additionally, the system should ensure that each book is associated with an existing author, and each borrower's contact information is provided. Prices for the books should always be non-negative, and it should be recorded whether a book is currently available for borrowing or not. How would you design the database schema while incorporating appropriate constraints to ensure data integrity?

Question:

Describe the database schema design for the library management system, considering the given requirements and incorporating the following constraints:

Ensure that the ISBN column in the `Books` table serves as a primary key to uniquely identify each book.

Implement a foreign key constraint to link the `AuthorID` column in the `Books` table to the `AuthorID` column in the `Authors` table, ensuring that each book is associated with an existing author.

Enforce a not null constraint on the `LibraryCardNumber` column in the `Borrowers` table to ensure that each borrower has a unique library card number.

Use a unique constraint to ensure that each borrower's `Email` address in the `Borrowers` table is unique.

Apply a check constraint to the `Price` column in the `Books` table to ensure that prices are always non-negative.

Utilize a default constraint to set the default value of the `Availability` column in the `Books` table to indicate whether a book is currently available for borrowing.

SQL SERVER CLAUSES

SQL SERVER CLAUSES

- SQL clause helps us to retrieve a set or bundles of records from the table.
- It helps us to specify a condition on the columns or the records of a table.
- A clause is a built-in function that helps to fetch the required records from a database table.
- A clause receives a conditional expression, i.e. a column name or some terms involving the columns.
- When large amount of data is stored in database, clauses are helpful to filter and analyze queries.



SQL SERVER CLAUSES

Different clauses available in the Structured Query Language are as follows:

- WHERE CLAUSE
- GROUP BY CLAUSE
- HAVING CLAUSE
- ORDER BY CLAUSE

WHERE Clause:

- The `WHERE` clause is used to filter rows before they are grouped or sorted.
- It applies conditions to individual rows of the table.
- It is used with the `SELECT`, `UPDATE`, and `DELETE` statements.
- Example: `SELECT * FROM Products WHERE Price > 100;`

GROUP BY Clause:

- The `GROUP BY` clause is used to group rows that have the same values into summary rows.
- It aggregates data based on specified columns and returns one row for each group.
- It is typically used with aggregate functions like `SUM`, `COUNT`, `AVG`, etc.
- Example: `SELECT Department, SUM(Salary) FROM Employees GROUP BY Department;`

HAVING Clause:

- The `HAVING` clause is used to filter the results of aggregate functions in a `GROUP BY` query.
- It applies conditions to the grouped rows after the `GROUP BY` operation has been performed.
- It is used to filter groups, not individual rows.
- **Example:** `SELECT Department, SUM(Salary) FROM Employees GROUP BY Department HAVING SUM(Salary) > 10000;`

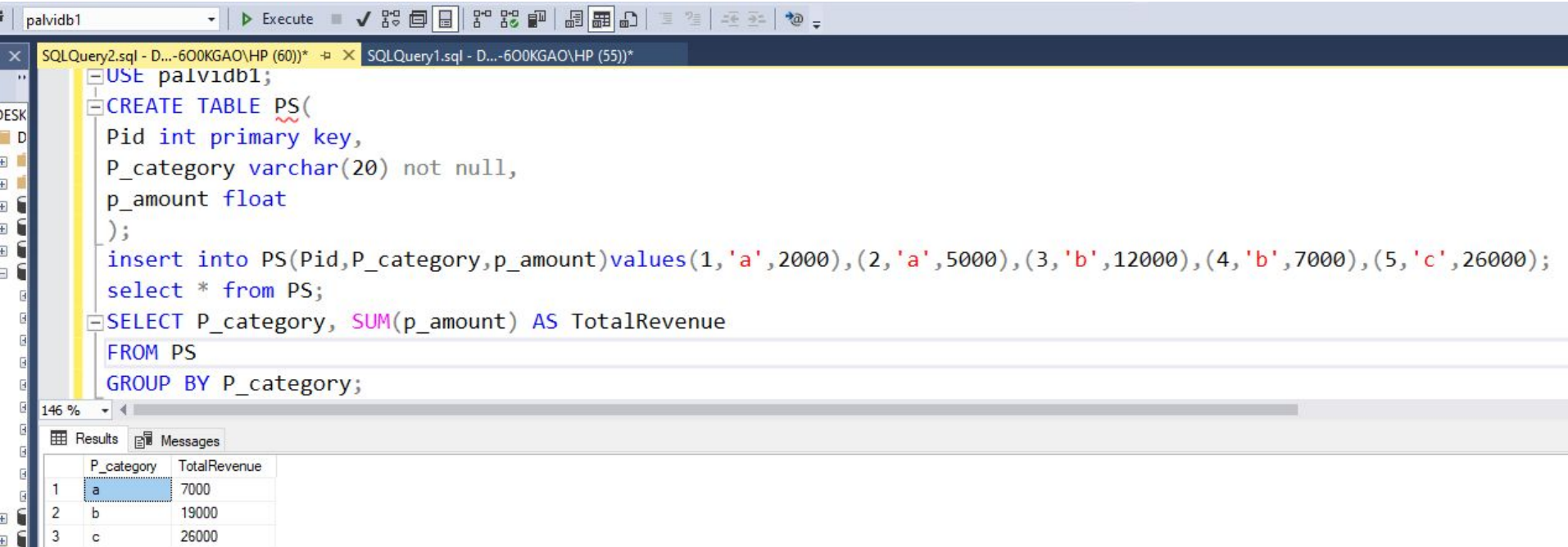
ORDER BY Clause:

- The `ORDER BY` clause is used to sort the result set based on specified columns.
- It sorts the rows returned by a query in ascending or descending order.
- It is used with the `SELECT` statement.
- **Example:** `SELECT * FROM Products ORDER BY Price DESC;`

In summary:

- `WHERE` is used to filter individual rows before grouping or sorting.
- `GROUP BY` is used to group rows with the same values into summary rows.
- `HAVING` is used to filter the grouped rows based on aggregate conditions.
- `ORDER BY` is used to sort the result set based on specified columns.

Group By Clause



The screenshot shows a SQL IDE interface. The top toolbar includes an 'Execute' button. The active query window is titled 'SQLQuery1.sql - D:\...-600KGAO\HP (55)*' and contains the following SQL code:

```
USE palvidb1;  
CREATE TABLE PS(  
    Pid int primary key,  
    P_category varchar(20) not null,  
    p_amount float  
);  
insert into PS(Pid,P_category,p_amount) values(1, 'a', 2000), (2, 'a', 5000), (3, 'b', 12000), (4, 'b', 7000), (5, 'c', 26000);  
select * from PS;  
SELECT P_category, SUM(p_amount) AS TotalRevenue  
FROM PS  
GROUP BY P_category;
```

The bottom panel shows the 'Results' tab with a table containing 3 rows and 2 columns: 'P_category' and 'TotalRevenue'.

	P_category	TotalRevenue
1	a	7000
2	b	19000
3	c	26000

Group By

```
USE palvidb1;
```

```
CREATE TABLE PS(
```

```
Pid int primary key,
```

```
P_category varchar(20) not null,
```

```
p_amount float
```

```
);
```

```
insert into PS(Pid,P_category,p_amount)values(1,'a',2000),(2,'a',5000),(3,'b',12000),(4,'b',7000),(5,'c',26000);
```

```
select * from PS;
```

```
SELECT P_category, SUM(p_amount) AS TotalRevenue
```

```
FROM PS
```

```
GROUP BY P_category;
```

Where Clause

```
SELECT P_category, SUM(p_amount) AS TotalRevenue  
FROM PS  
WHERE p_amount < 7000  
GROUP BY P_category;
```

146 %



Results



Messages

	P_category	TotalRevenue
1	a	7000

Tasks To Be done by students

Query to calculate total revenue for each category, filtered by revenue greater than \$600, and include categories with total revenue greater than \$1500

Order by

```
SELECT p_category, SUM(p_amount) AS TotalRevenue  
FROM PS  
WHERE p_amount > 600  
GROUP BY p_category  
HAVING SUM(p_amount) > 1500  
ORDER BY TotalRevenue DESC;
```



```
USE palvidb1;
```

```
CREATE TABLE employees (
```

```
    employee_id INT PRIMARY KEY,
```

```
    employee_name VARCHAR(50),
```

```
    department VARCHAR(50),
```

```
    hire_date DATE
```

```
);
```

```
-- Inserting sample data into the employees table
```

```
INSERT INTO employees (employee_id, employee_name, department, hire_date)
```

```
VALUES
```

```
(1, 'John Doe', 'Sales', '2020-01-15'),
```

```
(2, 'Jane Smith', 'Sales', '2019-05-20'),
```

```
(3, 'Michael Johnson', 'Marketing', '2020-03-10'),
```

```
(4, 'Emily Brown', 'Marketing', '2018-12-05'),
```

```
(5, 'David Lee', 'Sales', '2021-02-28'),
```

```
(6, 'Sarah Adams', 'HR', '2019-09-10');
```

ARITHMETIC OPERATORS

- Arithmetic operators perform simple arithmetic operations such as addition, subtraction, multiplication etc.

Operators	Description
+	Addition
-	Subtraction
*	Multiplication
/	Divide
%	Modulo (Remainder)

COMPARISON OPERATORS

- We can compare two values using comparison operators in SQL.
- These operators return either 1 (means true) or 0 (means false).

Operators	Description
=	Equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
<>, !=	Not equal to

LOGICAL OPERATORS

- We can use logical operators to compare multiple SQL commands
- These operators return either 1 (means true) or 0 (means false)

Logical operators available in SQL are,

- ANY and ALL
- AND, OR and NOT
- BETWEEN
- EXISTS
- IN
- LIKE
- IS NULL

Connect

DESKTOP-EI4423A (SQL Server 15.0.2)

- Databases
 - System Databases
 - Database Snapshots
 - CodeFirstDB
 - Courseradb
 - Database Diagrams
 - Tables
 - System Tables
 - FileTables
 - External Tables
 - Graph Tables
 - dbo.Departments
 - dbo.Employees
 - Views
 - External Resources
 - Synonyms
 - Programmability
 - Service Broker
 - Storage
 - Security
- DatabaseFirstDB
- productdb
- SalesOrdersDB
- SamoleDB

```
SELECT * FROM Departments;
```

```
SELECT * FROM Employees;
```

```
SELECT EmpName, EmpSalary, EmpSalary*12  
FROM Employees;
```

```
SELECT EmpName, EmpSalary, EmpSalary*12 AnnualSalary  
FROM Employees;
```

158 %

Results Messages

	EmpName	EmpSalary	AnnualSalary
1	King Kochhar	21000	252000
2	Sarah Bowling	32000	384000
3	John Smith	43000	516000
4	Srishti Sharma	30000	360000
5	Gautam Bhalla	10000	120000
6	Roger Lee	40000	480000
7	Sujatha Sharma	35000	420000
8	Tanya Gangwani	22000	264000

File Edit View Query Project Tools Window Help

New Query

Courseradb

Execute

Object Explorer

Connect

DESKTOP-EI4423A (SQL Server 15.0.2)

Databases

- System Databases
- Database Snapshots
- CodeFirstDB
- Courseradb
 - Database Diagrams
 - Tables
 - System Tables
 - FileTables
 - External Tables
 - Graph Tables
 - dbo.Departments
 - dbo.Employees
 - Views
 - External Resources
 - Synonyms
 - Programmability
 - Service Broker
 - Storage
 - Security
- DatabaseFirstDB
- productdb
- SalesOrdersDB

DESKTOP-EI4423A...b - dbo.Employees

CourseraScripts.sql...hawnaGunwani (62)*

```
SELECT EmpName, EmpSalary
FROM Employees
WHERE EmpSalary >= 30000;

SELECT EmpName, EmpSalary
FROM Employees
WHERE EmpName = 'Sarah Bowling';
```

158 %

Results Messages

	EmpName	EmpSalary
1	Sarah Bowling	32000
2	John Smith	43000
3	Srishti Sharma	30000
4	Roger Lee	40000
5	Sujatha Sharma	35000

Microsoft SQL Server Management Studio interface showing a query execution in the Courseradb database.

Object Explorer:

- DESKTOP-E14423A (SQL Server 15.0.2)
- Databases
 - System Databases
 - Database Snapshots
 - CodeFirstDB
 - Courseradb
 - Database Diagrams
 - Tables
 - System Tables
 - FileTables
 - External Tables
 - Graph Tables
 - dbo.Departments
 - dbo.Employees
 - Views
 - External Resources
 - Synonyms
 - Programmability
 - Service Broker
 - Storage
 - Security
 - DatabaseFirstDB
 - productdb
 - SalesOrdersDB
 - SampleDB

Query Editor:

```
SELECT EmpName, EmpSalary
FROM Employees
WHERE EmpName = 'Sarah Bowling' OR EmpName = 'Tanya Gangwani';

SELECT EmpName, EmpSalary
FROM Employees
WHERE EmpName = 'Sarah Bowling' AND EmpSalary > 32000;
```

Results:

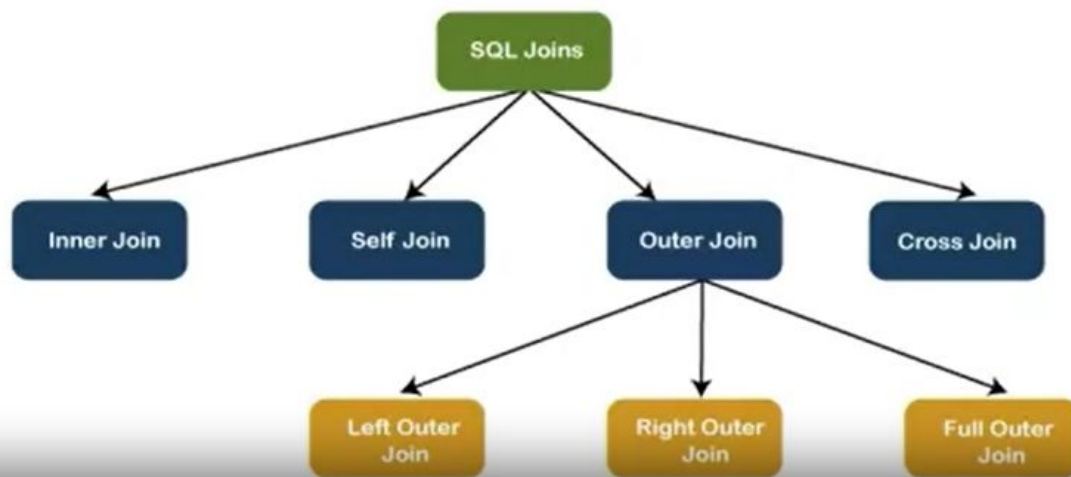
	EmpName	EmpSalary
1	Sarah Bowling	32000

Status Bar: Query executed successfully. DESKTOP-E14423A (15.0 RTM) | DESKTOP-E14423A\JhaenaGumwani | Courseradb | 00:00:00 | 1 rows

SQL SERVER JOINS

- Joins are used to combine rows from two or more tables based on related column between them.
- Joins are essential for retrieving data that is distributed across multiple tables.
- They allow us to establish relationships between tables.

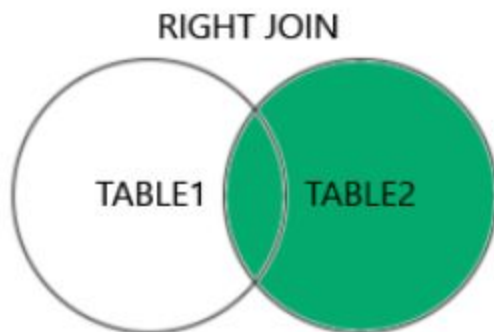
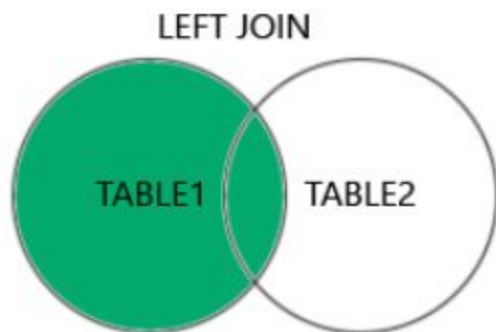
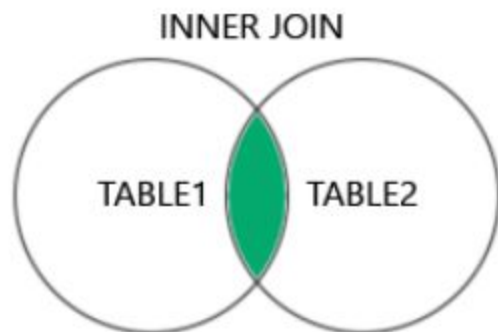
Types of Join



Different Types of SQL JOINS

Here are the different types of the JOINS in SQL:

- **(INNER) JOIN** : Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN** : Returns all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN** : Returns all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN** : Returns all records when there is a match in either left or right table



```
select * from TblCountry inner join state or
```

146 %

Results

Messages

	CountryId	CountryName
1	1	India
2	2	Nepal

	Sid	CountryId	Name
1	1	123	ma
2	2	1	goa
3	3	2	bihar
4	4	4	california

```

use learning;
select * from TblCountry;
select * from state;

select * from TblCountry inner join state on TblCountry.CountryId=state.CountryId;

```

146 %

Results Messages

	CountryId	CountryName	Sid	CountryId	Name
1	1	India	2	1	goa
2	2	Nepal	3	2	bihar

Tasks for students

Perform tasks on left,right, full join

SQL VIEWS

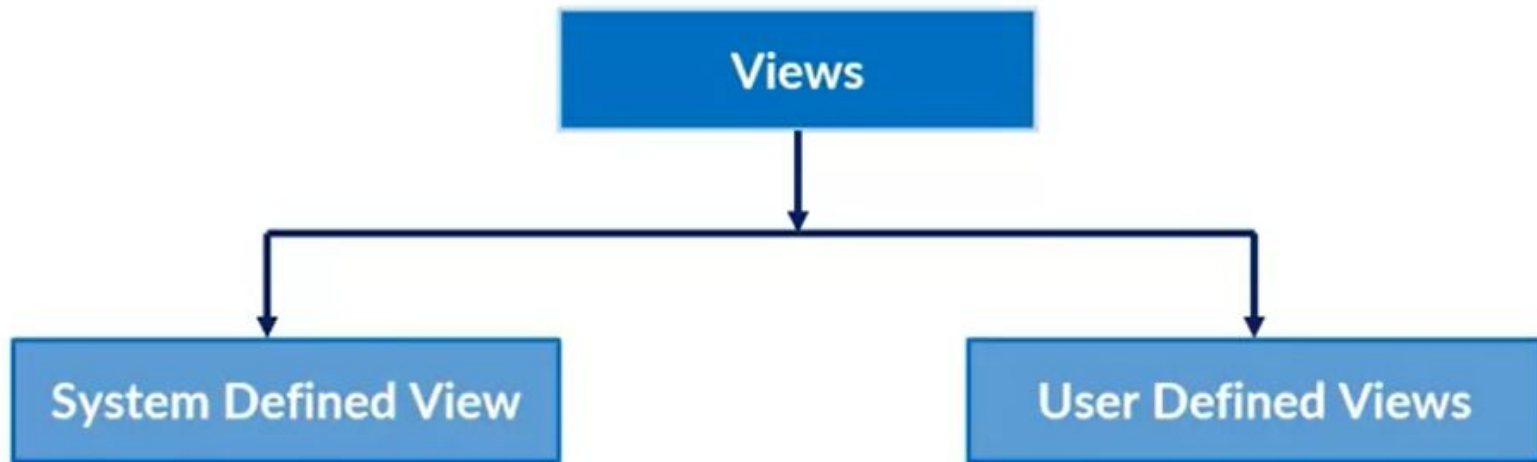
- A view is a database object that has no values.
- It is a virtual table, which is created according to the result set of an SQL query.
- However, it looks similar to an actual table containing rows and columns.
- Therefore, we can say that its contents are based on the base table.
- It is operated similarly to the base table but does not contain any data of its own.
- Its name is always unique, like tables.
- Views differ from tables as they are definitions that are created on top of other tables (or views).
- If any changes occur in the underlying table, the same changes reflected in the views also.

Views

Views in SQL are a kind of virtual table. A view also has rows and columns like tables, but a view doesn't store data on the disk like a table. View defines a customized query that retrieves data from one or more tables, and represents the data as if it was coming from a single source.

We can create a view by selecting fields from one or more tables present in the database. A View can either have all the rows of a table or specific rows based on certain conditions.

There are two types of Views:



SQL Server Enterprise Manager interface showing a query execution and results.

Object Explorer: Displays the database structure for 'DESKTOP-600KGAO\SQLEXPRESS (SQL Server 16.0.1000 - DESKTOP-6C)'. The 'Databases' folder is expanded, showing 'System Databases', 'Database Snapshots', 'CodeFirst1tdb', 'CodeFirstDb', 'DemoDb', 'Learning', and 'New1'. The 'New1' database is selected, showing 'Database Diagrams', 'Tables', 'Views', 'External Resources', 'Synonyms', 'Programmability', 'Query Store', 'Service Broker', 'Storage', and 'Security'. The 'Tables' folder is expanded, showing 'System Tables', 'FileTables', 'External Tables', 'Graph Tables', 'dbo.Demo', 'dbo.PTABLE', and 'dbo.Student'.

Query Editor: Shows the execution of a SQL query in 'SQLQuery1.sql - D:\...-600KGAO\HP (53))'. The query is:

```
USE new1;
select * from PTABLE;
Go
Create View MyView2 AS
SELECT SNo, SName
FROM PTABLE;
Go
SELECT * FROM MyView2;
```

Results: The query results are displayed in a table with 5 rows and 4 columns (Sno, Sname, SCountry). The first row is highlighted.

	Sno	Sname	SCountry
1	1	Palvi	Japan
2	2	Soni	Russia
3	3	Puneet	China
4	4	Sahil	America
5	5	Tushar	US

Messages: The 'Messages' tab is also visible, showing the execution of the query.

Complex view-create following tables

StudentDetails

S_ID	NAME	ADDRESS
1	Harsh	Kolkata
2	Ashish	Durgapur
3	Pratik	Delhi
4	Dhanraj	Bihar
5	Ram	Rajasthan

StudentMarks

ID	NAME	MARKS	AGE
1	Harsh	90	19
2	Suresh	50	20
3	Pratik	80	19
4	Dhanraj	95	21
5	Ram	85	18

Query

```
CREATE VIEW MarksView AS
```

```
SELECT StudentDetails.NAME, StudentDetails.ADDRESS, StudentMarks.MARKS
```

```
FROM StudentDetails, StudentMarks
```

```
WHERE StudentDetails.NAME = StudentMarks.NAME;
```

```
SELECT * FROM MarksView;
```

Output

NAME	ADDRESS	MARKS
Harsh	Kolkata	90
Pratik	Delhi	80
Dhanraj	Bihar	95
Ram	Rajasthan	85

Stored Procedure

Stored procedures are sets of SQL statements that are stored in the database and can be executed repeatedly by calling the procedure name. They offer several advantages over ad-hoc SQL statements, such as better performance, code reusability, modularity, and improved security. Here's an introduction to stored procedures and user-defined stored procedures in SQL:

A stored procedure is a named collection of SQL statements and procedural logic that performs a specific task or set of tasks in a database management system. Once created, it is stored in the database and can be invoked multiple times by applications or users.

Creation: Stored procedures are created using the `CREATE PROCEDURE` or `CREATE OR ALTER PROCEDURE` statement. The body of the procedure can contain SQL statements, control-of-flow statements like IF-ELSE, loops, error handling, and more.

Parameters: Stored procedures can accept input parameters, allowing them to be more flexible and reusable. Parameters can be of different types (IN, OUT, or INOUT) and can be passed values when the procedure is called.

Execution: Stored procedures are executed using the `EXECUTE` or `EXEC` statement, followed by the procedure name and any required parameters. They can also be called from within other stored procedures, functions, triggers, or applications.

Syntax

```
CREATE PROCEDURE procedure_name
```

```
AS
```

```
sql_statement
```

```
GO;
```

USER DEFINED STORED PROCEDURE WITH EXAMPLE

The screenshot displays the SQL Server Enterprise Manager interface. On the left, the 'Object Explorer' pane shows the database structure for 'DESKTOP-600KGAO\SQLEXPRESS (SQL Server 16.0.1000 - DESKTOP-6C)'. The 'Databases' folder is expanded, showing 'System Databases', 'Database Snapshots', and several user databases: 'CodeFirs1tDb', 'CodeFirstDb', 'DemoDb', 'Learning', and 'New1'. The 'Tables' folder under 'New1' is expanded, showing 'System Tables', 'FileTables', 'External Tables', 'Graph Tables', 'dbo.Demo', and 'dbo.PTABLE'. The 'Columns', 'Keys', and 'Constraints' folders are also visible.

The main window shows a SQL query script in 'SQLQuery2.sql'. The script contains the following T-SQL code:

```
select * from PTABLE;  
go  
  
CREATE PROCEDURE GetSerialNumber  
    @Sno INT  
AS  
BEGIN  
    SELECT * FROM PTABLE WHERE Sno = @Sno;  
END;  
  
EXEC GetSerialNumber @Sno = 1;
```

The 'Results' pane at the bottom shows the output of the query, which is a table with four columns: 'Sno', 'Sname', 'SCountry', and 'Country'. The first row of data is displayed:

Sno	Sname	SCountry	Country
1	1	Palvi	Japan

This will return the details of the employee with `sno.` equal to 1.

You can also call the stored procedure from within a SQL script or application code, passing the parameter value as needed.

Stored procedures offer a powerful way to encapsulate logic and promote code reuse in database applications. They can contain complex SQL queries, control-of-flow statements, error handling, and more, making them versatile tools for database developers.

For selecting all entries of the table through procedure

```
create procedure get_id2  
as  
begin  
select * from procedure_table  
end  
  
exec get_id2;
```