

Database Training with Microsoft SQL Server (DAY – 1) SQL PART - 1



Authorized & published by Summitworks Technologies Inc

Agenda Day 1

SQL PART – 1

- Database Fundamentals
 - What is a Database?
 - Types of database
 - Why to use Database
- Database Design
 - Introduction to database design
 - Keys and Constraints
 - Data Types
 - Fundamental building blocks:
 - Table, columns, constraints

Database Fundamentals

What is Data?

- In simple words data can be facts related to any object in consideration.
- For example your name, age, height, weight, etc are some data related to you.
- A picture , image , file , pdf etc can also be considered data.

What is Database?

- A database is a collection of information that is organized so that it can easily be accessed, managed, and updated.
- You can query data in a database in a very efficient way
- You can relate data from different tables using JOINS.
- Your data stored using a built-in structure.
- Databases are ACID(Atomicity, Consistency, Isolation, Durability)



Benefits of Database

Why to use a database?

- Reduce the amount of time you spend managing data
- Analyze data in a variety of ways
- Standards can be enforced
- Security restrictions can be applied
- Improve the quality and consistency of information
- Eliminate or reduce Data Redundancy
- Data Integrity is maintained



Types of Database

Different types of databases:

- Flat file
 - file containing records that have no structured interrelationship
 - Text files, Comma/Tab Delimited Files
- XML
 - EXtensible Markup Language
 - Designed to be both human- and machine-readable
- Excel/Spreadsheets
 - Stores data in a tabular form with relationship
- Relational database
 - A database structured to recognize relations among stored items of information.

Database Fundamentals

What is Database Management System (DBMS)?

- Database Management System (DBMS) is a collection of programs which enables its users to access database, manipulate data, reporting / representation of data.
- We have four major types of DBMSs namely Hierarchical, Network, Relational, Object Oriented
- The most widely used DBMS is the relational model or RDBMSs that saves data in table formats. It uses SQL as the standard query language
- SQL language is used to query a database
- The database approach has many advantages when it comes to storing data compared to the traditional flat file based systems



Database Fundamentals

What is SQL?

- Structured Query Language (SQL) is the standard language for data manipulation in a DBMS. In simple words it's used to talk to the data in a DBMS. Following are types of SQL Statements
- Data Definition Language (DDL) allows you to create objects like Schemas, Tables in the database
- Data Control Language (DCL) allows you to manipulate and manage access rights on database objects
- Data Manipulation Language (DML) is used for searching, inserting, updating, and deleting data, which will be partially covered in this programming tutorial.

Database Fundamentals

Why it makes sense to learn SQL after NOSQL ?

- Relational databases have the following advantages over NOSQL databases;
- SQL(relational) databases have a mature data storage and management model . This is crucial for enterprise users.
- SQL databases support the notion of views which allow users to only see data that they are authorized to view. The data that they are not authorized to see is kept hidden from them.
- SQL databases support stored procedure sql which allow database developers to implement part of the business logic into the database.
- SQL databases have better security models compared to NoSQL databases.

What is a Key?

Keys are fields in a table which participate in below activities in RDBMS systems:

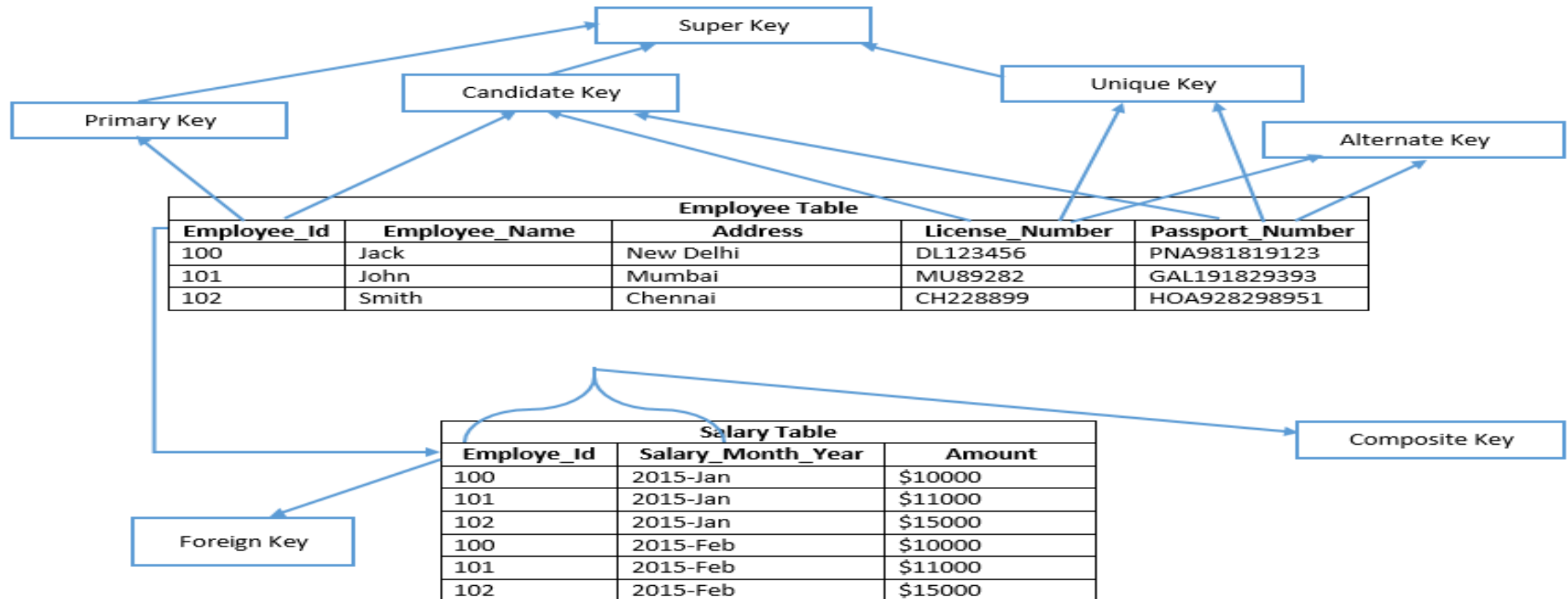
- To create relationships between two tables.
- To maintain uniqueness in a table.
- To keep consistent and valid data in database.
- Might help in fast data retrieval by facilitating indexes on column(s).

SQL Server supports various types of keys, which are listed below:

- Primary Key
- Unique Key
- Candidate Key
- Alternate Key
- Composite Key
- Super Key
- Foreign Key

Types of Keys

Before discussing each type in brief, have a look on the below image used as an example to define types of keys.



Types of keys

Candidate Key

Candidate Key:

A Candidate Key is a set of one or more fields/columns that can identify a record uniquely in a table.

There can be multiple Candidate Keys in one table.

Each Candidate Key can work as Primary Key.

- **Example:** Employee_Id, License_Number and Passport_Number are candidate keys

Primary Key

Primary Key:

A primary key is a single column value used to identify a database record uniquely.

It can not accept null, and duplicate values.

Only one Candidate Key can be Primary Key.

By default Primary key creates a clustered index

A table can have only one primary key

Example: Employee_Id is a primary key of Employee table.

Unique Key

Unique Key:

- Unique key is similar to primary key and does not allow duplicate values in the column.

It has below differences in comparison of primary key:

- It allows one null value in the column.
- By default, it creates a non clustered index

Alternate Key:

- Alternate key is a candidate key, currently not selected as primary key of the table.

Example: License_Number and Passport_Number are alternate keys.

Super Key

Super Key:

- Super key is a set of one or more than one keys that can be used to identify a record uniquely in a table. **Example:** Primary key, Unique key, Alternate key are a subset of Super Keys.

For example:

- {Employee_Id}, {Employee_Id, Employee_Name}, {Employee_Id, Employee_Name, Address} etc.
- License_Number and Passport_Number columns can also identify any row of the table uniquely. Any set of column which contains License_Number or Passport_Number or Employee_Id is a super key of the table.

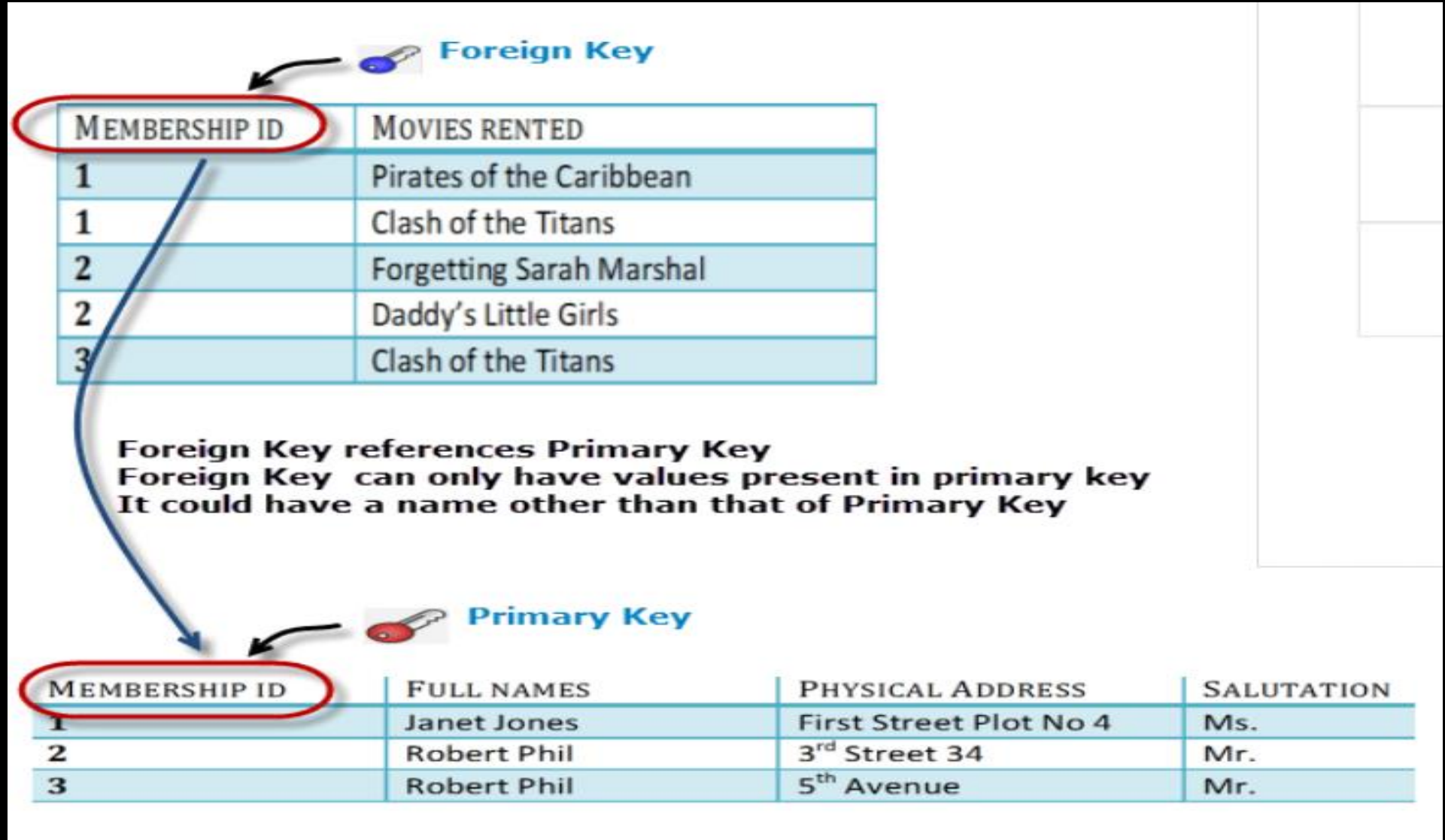
Composite Key

- Composite key (also known as compound key or concatenated key) is a combination of two or more columns that helps to identify each row of a table uniquely.
- Individual column of composite key might not be able to uniquely identify the record. It can be a primary key or candidate key also.

Example:

- In salary table, Employee_Id and Salary_Month_Year are combined together to identify each row uniquely in Salary table.

Foreign Key



Foreign Key



- Foreign Key references the primary key of another Table! It helps connect your Tables
- A foreign key can have a different name from its primary key
- It ensures rows in one table have corresponding rows in another
- Unlike the Primary key, they do not have to be unique. Most often they aren't
- Foreign keys can be null even though primary keys can not

Foreign Key

Why do you need a foreign key?

You will only be able to insert values into your foreign key that exist in the Primary key in the parent table. This helps in referential integrity.

Insert a record in Table 2 where Member ID = 101

MEMBERSHIP ID	MOVIES RENTED
101	Mission Impossible

But Membership ID 101 is not present in Table 1

MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION
1	Janet Jones	First Street Plot No 4	Ms.
2	Robert Phil	3 rd Street 34	Mr.
3	Robert Phil	5 th Avenue	Mr.

Database will throw an **ERROR**. This helps in referential integrity


- The above problem can be overcome by declaring membership id from Table 2 as foreign key of membership id from Table 1
- Now, if somebody tries to insert a value in the membership id field that does not exist in the parent table, an error will be shown!

Referential integrity

What is Referential integrity?

Referential integrity (RI) is a relational **database** concept, which states that table relationships must always be consistent. In other words, Referential integrity means that the foreign key in any referencing table must always refer to a valid row in the referenced table. Referential integrity ensures that the relationship between two tables remains synchronized during updates and deletes.

DEPT		EMP		
DEPTNO	DNAME	EMPNO	DEPTNO	ENAME
10	ACCOUNTS	7782	10	CLARK
20	RESEARCH	7934	10	MILLER
30	SALES	7876	20	ADAMS
		7902	20	FORD
		7900	30	JAMES

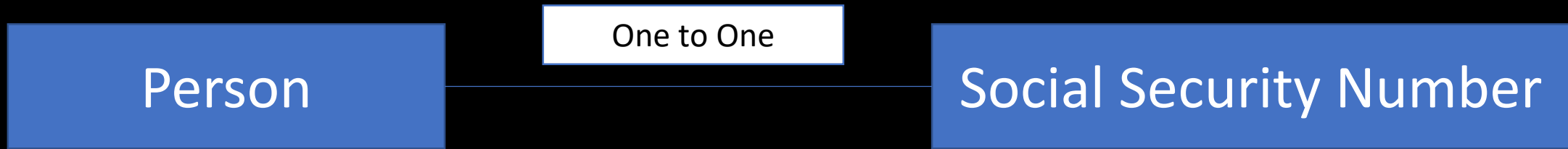


Database Relationship

Logical relationship

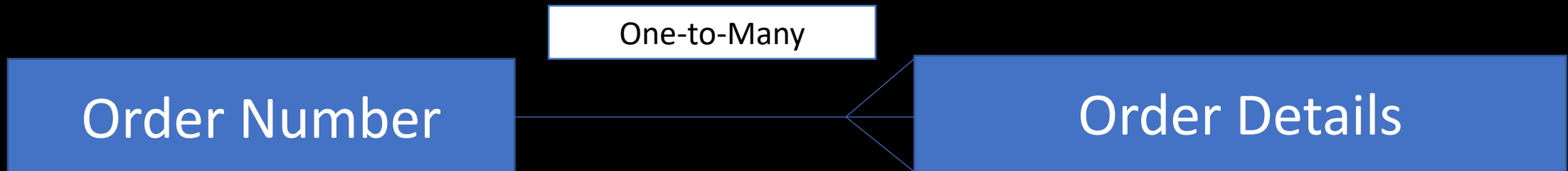
- **One to One**

- Both tables can have only one record on either side of the relationship
- Each primary key value relates to only one (or no) record in the related table



- **One-to-many**

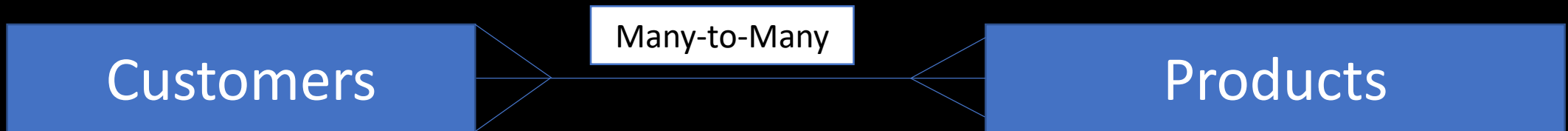
- The primary key table contains only one record that relates to none, one, or many records in the related table.



Database Relationship

- **Many to Many**

- Each record in both tables can relate to any number of records (or no records) in the other table.
- **Relational databases do not support many-to-many relationships. We need to introduce a junction entity**



Data Types

Data types define what type of data a column can contain.

Properly defining the fields in a table is important to the overall optimization of your database.

While creating any table or variable, in addition to specifying the name, you also set the **Type of Data** it will store.

MS SQL server support following categories of Data type:

-
- Date and time
- Numeric
- Character strings
- Unicode character strings
- Binary strings

Date and Time

It stores data of type date and time

Date and Time Data Type					
Data Type	Description	Storage size	Accuracy	Lower Range	Upper Range
DateTime	Used for specifying a date and time from January 1, 1753 to December 31, 9999. It has an accuracy of 3.33 milliseconds.	8 bytes	Rounded to increments of .000, .003, .007	1753-01-01	9999-12-31
smalldatetime	Used for specifying a date and time from January 1, 0001 to December 31, 9999. It has an accuracy of 100 nanoseconds	4 bytes, fixed	1 minute	1900-01-01	2079-06-06
date	Used to store only date from January 1, 0001 to December 31, 9999	3 bytes, fixed	1 day	0001-01-01	9999-12-31
time	Used for storing only time only values with an accuracy of 100 nanoseconds.	5 bytes	100 nanoseconds	00:00:00.0000000	23:59:59.9999999
datetimeoffset	Similar to datetime but has a time zone offset	10 bytes	100 nanoseconds	0001-01-01	9999-12-31
datetime2	Used for specifying a date and time from January 1, 0001 to December 31, 9999	6 bytes	100 nanoseconds	0001-01-01	9999-12-31

Numeric

Numeric has nine types of sub-data types

Exact Numeric Data Types				
Data Type	Description	Lower limit	Upper limit	Memory
bigint	It stores whole numbers in the range given	-2^{63} (−9,223,372,036,854,775,808)	$2^{63}-1$ (9,223,372,036,854,775,807)	8 bytes
int	It stores whole numbers in the range given	-2^{31} (−2,147,483,648)	$2^{31}-1$ (2,147,483,647)	4 bytes
smallint	It stores whole numbers in the range given	-2^{15} (−32,768)	2^{15} (32,767)	2 bytes
tinyint	It stores whole numbers in the range given	0	255	1 byte
bit	It can take 0, 1, or NULL values.	0	1	1 byte/8bit column
decimal	Used for scale and fixed precision numbers	$-10^{38}+1$	$10^{38}-1$	5 to 17 bytes
numeric	Used for scale and fixed precision numbers	$-10^{38}+1$	$10^{38}-1$	5 to 17 bytes
money	Used monetary data	−922,337,203,685,477.5808	+922,337,203,685,477.5807	8 bytes
smallmoney	Used monetary data	−214,478.3648	+214,478.3647	4 bytes

Character Strings

This category is related to a character type. It allows the user to define the data type of character which can be of fixed and variable length. It has four kinds of data types.

Character Strings Data Types

Data Type	Description	Lower limit	Upper limit	Memory
char	It is a character string with a fixed width. It stores a maximum of 8,000 characters.	0 chars	8000 chars	n bytes
varchar	This is a character string with variable width	0 chars	8000 chars	n bytes + 2 bytes
varchar (max)	This is a character string with a variable width. It stores a maximum of 1,073,741,824 characters.	0 chars	2 ³¹ chars	n bytes + 2 bytes
text	This is a character string with a variable width. It stores a maximum 2GB of text data.	0 chars	2,147,483,647 chars	n bytes + 4 bytes

Unicode Character Strings

This category store the full range of Unicode character which uses the UTF-16 character encoding.

Unicode Character String Data Types

Data Type	Description	Lower limit	Upper limit	Memory
nchar	It is a Unicode string of fixed width	0 chars	4000 chars	2 times n bytes
nvarchar	It is a unicode string of variable width	0 chars	4000 chars	2 times n bytes + 2 bytes
ntext	It is a unicode string of variable width	0 chars	1,073,741,823 char	2 times the string length

Variables

What is Variable?

In MS SQL, variables are the object which acts as a placeholder to a memory location.

Variable hold single data value.

Types of Variable:

MS SQL has two types of variables:

- Local variable
- Global variable

Types of Variables

Local variable:

- A user declares the local variable
- By default, a local variable starts with @
- Every local variable scope has the restriction to the **current batch or procedure** within any given session

Global variable:

- The system maintains the global variable
- A user cannot declare them
- The global variable starts with @@
- It stores **session related information**

Below figure explain two types of variable available in MS SQL server.

@ User Defined



Local
Variable

© guru99.com

@@System Defined



Global
Variable

Type of Variables in SQL Server

Declaring variables

Examples of Declaring a variable:

Query: With 'AS'

```
DECLARE @COURSE_ID AS INT;
```

Query: Without 'AS'

```
DECLARE @COURSE_NAME VARCHAR (10);
```

Query: DECLARE two variables

```
DECLARE @COURSE_ID AS INT, @COURSE_NAME VARCHAR (10);
```

Assigning a value to a VARIABLE

We can assign values to a variable as follows:

1. During variable declaration using DECLARE keyword

Example:

Query:

```
DECLARE @COURSE_ID AS INT = 5  
PRINT @COURSE_ID
```

2. Using SET - When we want to keep declaration and initialization separate. SET can be used to assign values to the variable, post declaring a variable. One SET Keyword can be used to assign a value to only **one variable**.

Example:

```
DECLARE @COURSE_ID AS INT  
SET @COURSE_ID = 5  
PRINT @COURSE_ID
```

```
DECLARE @Local_Variable_1 <Data_Type>, @Local_Variable_2 <Data_Type>,  
SET @Local_Variable_1 = <Value_1>  
SET @Local_Variable_2 = <Value_2>
```

Database Constraints

SQL constraints are used to specify rules for the data in a table.

If there is any violation between the constraint and the data action, the action is aborted by the constraint.

Constraints can be specified when the table is created (inside the CREATE TABLE statement) or after the table is created (inside the ALTER TABLE statement).

Constraints can be defined in two ways

- 1. The constraints can be specified immediately after the column definition. This is called column-level definition.
- 2. The constraints can be specified after all the columns are defined. This is called table-level definition.

Database Constraints

There are 7 types of constraints that are grouped in 4 categories:

Entity Integrity:

1. Primary Key -- Identifies a row uniquely which will not allow NULL values and Duplicate Values
2. Unique Key -- Identifies a row uniquely which will allow NULL values

Referential Integrity:

3. Foreign Key -- References primary key of same table or another table

Domain Integrity:

4. Default -- Allow to set default value
5. Check -- Defines a logic on a column
6. Not Null -- Enforces the column must contain a value

Primary Key Constraint

A primary key is a column or a group of columns that uniquely identifies each row in a table. We can create a primary key for a table by using the PRIMARY KEY CONSTRAINT. If the primary key consists of only one column, we can define Column level constraint. Else if the primary key has two or more columns, we define Table level constraint. SQL Server also automatically creates a unique clustered index when you create a primary key.

Example:

Column level constraint

```
CREATE TABLE sales.activities (  
    activity_id INT PRIMARY KEY IDENTITY,  
    activity_name VARCHAR (255) NOT NULL,  
    activity_date DATE NOT NULL  
);
```

Table level constraint

```
CREATE TABLE sales.participants(  
    activity_id int,  
    customer_id int,  
    PRIMARY KEY(activity_id, customer_id)  
);
```

Primary Key Constraint

We can add the PRIMARY KEY Constraint to an existing table using the ALTER Table Statement.

Example:

```
CREATE TABLE sales.events(  
    event_id INT NOT NULL,  
    event_name VARCHAR(255),  
    start_date DATE NOT NULL,  
    duration DEC(5,2)  
);
```

To make the event_id column as the primary key, you use the following ALTER TABLE Statement.

```
ALTER TABLE sales.events  
ADD PRIMARY KEY(event_id);
```

Unique Key Constraint

UNIQUE constraints allow you to ensure that the data stored in a column, or a group of columns, is unique among the rows in a table.

Example:

```
CREATE SCHEMA hr;  
GO  
  
CREATE TABLE hr.persons(  
    person_id INT IDENTITY PRIMARY KEY,  
    first_name VARCHAR(255) NOT NULL,  
    last_name VARCHAR(255) NOT NULL,  
    email VARCHAR(255) UNIQUE  
);
```

Insert a new row into the hr.person table

```
INSERT INTO hr.persons(first_name, last_name, email)  
VALUES('John', 'Doe', 'j.doe@bike.stores');
```

If you attempt to insert a duplicate row, SQL Server rejects the change and returns an error message stating that the UNIQUE constraint has been violated.

```
Messages  
Msg 2627, Level 14, State 1, Line 11  
Violation of UNIQUE KEY constraint 'UQ_persons_AB6E61640EF5EB6F'. Cannot insert duplicate key in object 'hr.persons'. The duplicate key value  
The statement has been terminated.
```

Unique Key Constraint

UNIQUE constraint accepts NULL since it treats Null as regular values, it only allows one Null per column.

Example:

The following statement inserts a row whose value in the email column is NULL

```
INSERT INTO hr.persons(first_name, last_name)
VALUES('John','Smith');
```

If we try to insert one more NULL into the email column we will get an error

Messages

Msg 2627, Level 14, State 1, Line 11

Violation of UNIQUE KEY constraint 'UQ__persons__AB6E61640EF5EB6F'. Cannot insert duplicate key in object 'hr.persons'.
The statement has been terminated.

Foreign Key Constraints

FOREIGN KEY

- A foreign key is a column or a group of columns in one table that uniquely identifies a row of another table (or the same table in case of self-reference).
- It establishes a relationship between two columns in the same table or between different tables.
- For a column to be defined as a Foreign Key, it should be defined as a Primary Key in the table which it is referring.
- One or more columns can be defined as Foreign key.

```
CREATE TABLE vendors (  
    vendor_id INT IDENTITY PRIMARY KEY,  
    vendor_name VARCHAR(100) NOT NULL,  
    group_id INT NOT NULL,  
    CONSTRAINT fk_group FOREIGN KEY (group_id)  
    REFERENCES vendor_groups(group_id)  
);
```

Foreign Key Constraints

The parent table is the table to which the foreign key constraint references and the child table is the one to which the foreign key constraint is applied.

Parent table

```
CREATE TABLE vendor_groups (  
  group_id INT IDENTITY PRIMARY KEY,  
  group_name VARCHAR (100) NOT NULL  
);
```

Child table

```
CREATE TABLE vendors (  
  vendor_id INT IDENTITY PRIMARY KEY,  
  vendor_name VARCHAR(100) NOT NULL,  
  group_id INT NOT NULL,  
  CONSTRAINT fk_group FOREIGN KEY (group_id)  
  REFERENCES vendor_groups(group_id)  
);
```

Insert some rows in the vendor_groups table (Parent table)

```
INSERT INTO vendor_groups(group_name)VALUES('Third-Party Vendors'),('Interco Vendors'),('One-time Vendors');
```

Insert a new vendor with a vendor group_id into the vendor table (Child table)

```
INSERT INTO vendors(vendor_name, group_id)VALUES('ABC Corp',1);
```

Insert a new vendor whose group_id does not exist in the vendor groups table

```
INSERT INTO vendors(vendor_name, group_id)VALUES('XYZ Corp',4);
```

Msg 547, Level 16, State 0, Line 10

The INSERT statement conflicted with the FOREIGN KEY constraint "fk_group".
The statement has been terminated.

NOT NULL Constraints

The NOT NULL constraint enforces a field to always contain a value. This means that you cannot insert a new record, or update a record without adding a value to this field. Which means the column will not allow Null values.

NOT NULL constraints are always written as column constraints.

```
CREATE TABLE persons(  
  person_id INT IDENTITY PRIMARY KEY,  
  first_name VARCHAR(255) NOT NULL,  
  last_name VARCHAR(255) NOT NULL,  
  email VARCHAR(255) NOT NULL,  
  phone VARCHAR(20)  
);
```

Check Constraints

- This constraint defines a business rule on a column. All the rows must satisfy this rule. The constraint can be applied for a single column or a group of columns.
- The CHECK constraint is used to limit the value range that can be placed in a column

Example: We use the CHECK constraint to allow only positive unit price

```
CREATE TABLE products(  
    product_id INT IDENTITY PRIMARY KEY,  
    product_name VARCHAR(255) NOT NULL,  
    unit_price DEC(10,2) CHECK(unit_price > 0)  
);
```

Insert a record to the products table with unit price is equal to 0

```
INSERT INTO products(product_name, unit_price)VALUES ('Awesome Free Bike', 0);
```

We will get this error msg

```
Msg 547, Level 16, State 0, Line 10  
The INSERT statement conflicted with the CHECK constraint "CK__products__unit_p__5165187F".  
The statement has been terminated.
```


Default Constraints

DEFAULT CONSTRAINT

- Specifies a default value for a column
- The DEFAULT constraint is used to insert a default value into a column. The default value will be added to all new records, if no other value is specified.

```
CREATE TABLE Persons
(
  P_Id int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Address varchar(255),
  City varchar(255) DEFAULT 'Sandnes'
)
```

```
CREATE TABLE Orders
(
  O_Id int NOT NULL,
  OrderNo int NOT NULL,
  P_Id int,
  OrderDate date DEFAULT GETDATE()
)
```

Default Constraints

Table Level Constraint:

```
CREATE TABLE Orders
(  
  O_Id int NOT NULL,  
  OrderNo int NOT NULL,  
  P_Id int,  
  OrderDate date,  
  DEFAULT GETDATE() FOR Orderdate
```