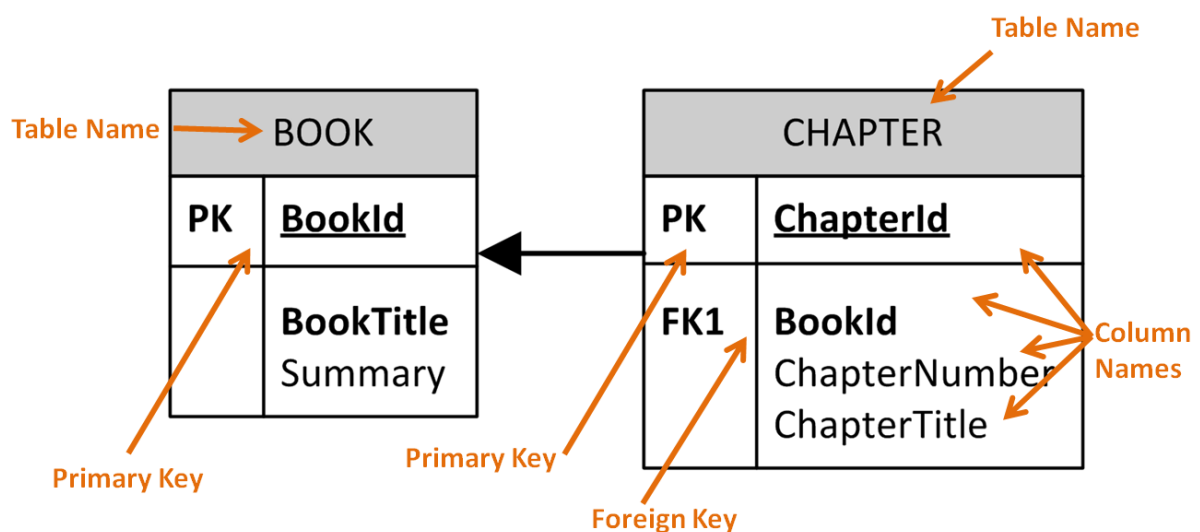


# Introduction to Database Systems

Hans-Petter Halvorsen



# Introduction to Database Systems

Hans-Petter Halvorsen

Copyright © 2017

E-Mail: [hans.p.halvorsen@usn.no](mailto:hans.p.halvorsen@usn.no)

Web: <https://www.halvorsen.blog>



<https://www.halvorsen.blog>

# Preface

This document explains the basic concepts of a database system and how to communicate with a database system.

The main focus in this document is on relational databases and Microsoft SQL Server.

For more information about Databases, visit my Blog:

<https://www.halvorsen.blog>

Some text in this document is based on text from [www.wikipedia.org](http://www.wikipedia.org).

# Table of Contents

Preface .....	3
Table of Contents .....	iv
1 Database Systems .....	1
1.1 RDBMS Components .....	1
1.2 Data warehouse .....	2
1.3 Relational Database.....	2
1.4 Real-time databases .....	2
1.5 NoSQL Databases.....	3
1.6 Database Management Systems .....	3
1.7 MDAC.....	4
1.7.1 ODBC.....	4
1.7.2 OLE DB .....	4
1.7.3 ADO (ActiveX Data Objects).....	4
2 Relational Databases .....	5
2.1 Tables .....	5
2.2 Unique Keys and Primary Key.....	6
2.3 Foreign Key .....	7
2.4 Views .....	8
2.5 Functions .....	8
2.6 Stored Procedures .....	9
2.7 Triggers .....	9
3 Structured Query Language (SQL) .....	10

---

3.1	Queries .....	10
3.2	Data Manipulation.....	11
3.3	Data Definition .....	12
3.4	Data Types .....	13
3.4.1	Character Strings.....	13
3.4.2	Bit Strings .....	13
3.4.3	Numbers .....	13
3.4.4	Date and Time.....	13
4	Database Modelling .....	14
4.1	ER Diagram .....	14
4.2	Microsoft Visio.....	15
4.3	ERwin .....	17
5	Microsoft SQL Server .....	19
5.1	Introduction.....	19
5.2	SQL Server Express .....	19
5.3	SQL Server Management Studio.....	20
5.4	Create a New Database .....	20
5.5	Backup/Restore .....	22
6	Microsoft Office Access.....	24
6.1	Introduction.....	24
6.2	Example Database .....	24
7	Creating and Using Tables .....	27
8	Creating and Using Views.....	30
9	Creating and using Stored Procedures .....	31
10	Creating and Using Triggers .....	33
11	Creating and Using Functions.....	35



# 1 Database Systems

A database is an integrated collection of logically related records or files consolidated into a common pool that provides data for one or more multiple uses.

One way of classifying databases involves the type of content, for example: bibliographic, full-text, numeric, and image. Other classification methods start from examining database models or database architectures.

The data in a database is organized according to a database model. The relational model is the most common.

A Database Management System (**DBMS**) consists of software that organizes the storage of data. A DBMS controls the creation, maintenance, and use of the database storage structures of organizations and of their end users. It allows organizations to place control of organization-wide database development in the hands of Database Administrators (DBAs) and other specialists. In large systems, a DBMS allows users and other software to store and retrieve data in a structured way.

Database management systems are usually categorized according to the database model that they support, such as the network, relational or object model. The model tends to determine the query languages that are available to access the database. One commonly used query language for the relational database is SQL, although SQL syntax and function can vary from one DBMS to another. A great deal of the internal engineering of a DBMS is independent of the data model, and is concerned with managing factors such as performance, concurrency, integrity, and recovery from hardware failures. In these areas there are large differences between products.

## 1.1 RDBMS Components

A Relational Database Management System (DBMS) consists of the following components:

- **Interface drivers** - A user or application program initiates either schema modification or content modification. These drivers are built on top of SQL. They provide methods to prepare statements, execute statements, fetch results, etc. An important example is the ODBC driver.

- **SQL engine** - This component interprets and executes the SQL query. It comprises three major components (compiler, optimizer, and execution engine).
- **Transaction engine** - Transactions are sequences of operations that read or write database elements, which are grouped together.
- **Relational engine** - Relational objects such as Table, Index, and Referential integrity constraints are implemented in this component.
- **Storage engine** - This component stores and retrieves data records. It also provides a mechanism to store metadata and control information such as undo logs, redo logs, lock tables, etc.

## 1.2 Data warehouse

A data warehouse stores data from current and previous years — data extracted from the various operational databases of an organization. It becomes the central source of data that has been screened, edited, standardized and integrated so that it can be used by managers and other end-user professionals throughout an organization.

## 1.3 Relational Database

A relational database matches data using common characteristics found within the data set. The resulting groups of data are organized and are much easier for people to understand.

For example, a data set containing all the real-estate transactions in a town can be grouped by the year the transaction occurred; or it can be grouped by the sale price of the transaction; or it can be grouped by the buyer's last name; and so on.

Such a grouping uses the relational model (a technical term for this is schema). Hence, such a database is called a "relational database."

The software used to do this grouping is called a relational database management system. The term "relational database" often refers to this type of software.

Relational databases are currently the predominant choice in storing financial records, manufacturing and logistical information, personnel data and much more.

Strictly, a relational database is a collection of relations (frequently called tables).

## 1.4 Real-time databases



A real-time database is a processing system designed to handle workloads whose state may change constantly. This differs from traditional databases containing persistent data, mostly unaffected by time. For example, a stock market changes rapidly and dynamically. Real-time processing means that a transaction is processed fast enough for the result to come back and be acted on right away. Real-time databases are useful for accounting, banking, law, medical records, multi-media, process control, reservation systems, and scientific data analysis. As computers increase in power and can store more data, real-time databases become integrated into society and are employed in many applications

## 1.5 NoSQL Databases

The next generation of database systems is known as NoSQL databases and document-oriented databases. NoSQL databases are often very fast, do not require fixed table schemas.

Examples of NoSQL systems: MongoDB and Oracle NoSQL Database.



## 1.6 Database Management Systems

There are Database Management Systems (DBMS), such as:

- Microsoft SQL Server
- Oracle
- Sybase
- dBase
- Microsoft Access
- MySQL from Sun Microsystems (Oracle)
- DB2 from IBM
- etc.

This document will focus on Microsoft Access and Microsoft SQL Server.

## 1.7 MDAC

The Microsoft Data Access Components (**MDAC**) is the framework that makes it possible to connect and communicate with the database. MDAC includes the following components:

- **ODBC** (Open Database Connectivity)
- **OLE DB**
- **ADO** (ActiveX Data Objects)

MDAC also installs several data providers you can use to open a connection to a specific data source, such as an MS Access database.

### 1.7.1 ODBC

**Open Database Connectivity (ODBC)** is a native interface that is accessed through a programming language that can make calls into a native library. In MDAC this interface is defined as a DLL. A separate module or driver is needed for each database that must be accessed.

### 1.7.2 OLE DB

**OLE** allows MDAC applications access to different types of data stores in a uniform manner. Microsoft has used this technology to separate the application from the data store that it needs to access. This was done because different applications need access to different types and sources of data, and do not necessarily need to know how to access technology-specific functionality. The technology is conceptually divided into consumers and providers. The consumers are the applications that need access to the data, and the provider is the software component that exposes an OLE DB interface through the use of the Component Object Model (or COM).

### 1.7.3 ADO (ActiveX Data Objects)

**ActiveX Data Objects (ADO)** is a high level programming interface to OLE DB. It uses a hierarchical object model to allow applications to programmatically create, retrieve, update and delete data from sources supported by OLE DB. ADO consists of a series of hierarchical COM-based objects and collections, an object that acts as a container of many other objects. A programmer can directly access ADO objects to manipulate data, or can send an SQL query to the database via several ADO mechanisms.

# 2 Relational Databases

A relational database matches data using common characteristics found within the data set. The resulting groups of data are organized and are much easier for people to understand.

For example, a data set containing all the real-estate transactions in a town can be grouped by the year the transaction occurred; or it can be grouped by the sale price of the transaction; or it can be grouped by the buyer's last name; and so on.

Such a grouping uses the relational model (a technical term for this is schema). Hence, such a database is called a "relational database."

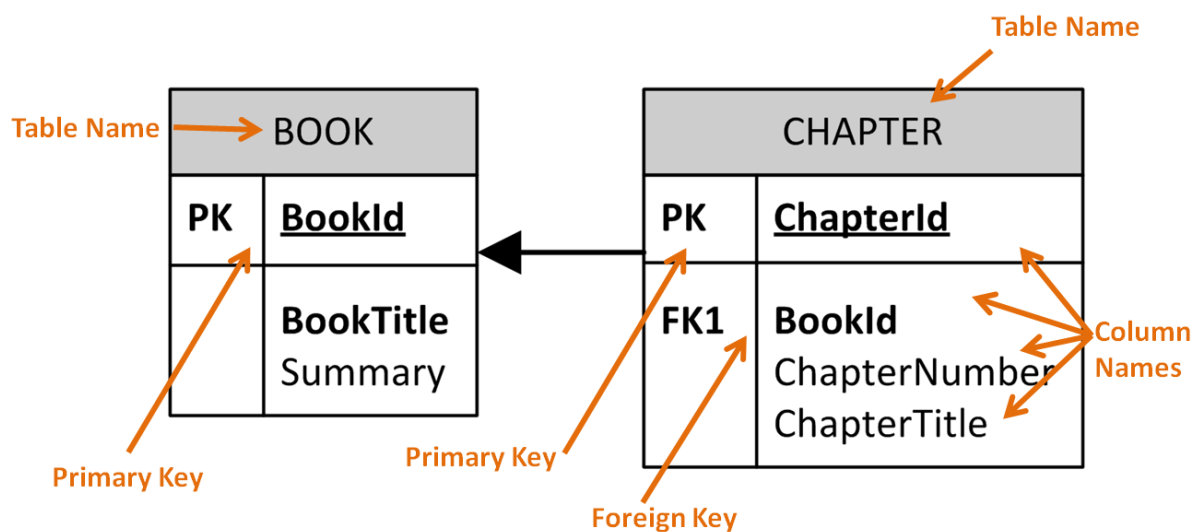
The software used to do this grouping is called a relational database management system. The term "relational database" often refers to this type of software.

Relational databases are currently the predominant choice in storing financial records, manufacturing and logistical information, personnel data and much more.

## 2.1 Tables

The basic units in a database are tables and the relationship between them. Strictly, a relational database is a collection of relations (frequently called tables).

Below we see how a relationship between two tables are defined using Primary Keys and Foreign Keys.



## 2.2 Unique Keys and Primary Key

In relational database design, a **unique key** or primary key is a candidate key to uniquely identify each row in a table. A unique key or primary key comprises a single column or set of columns. No two distinct rows in a table can have the same value (or combination of values) in those columns. Depending on its design, a table may have arbitrarily many unique keys but at most one primary key.

A unique key must uniquely identify all possible rows that exist in a table and not only the currently existing rows. Examples of unique keys are Social Security numbers or ISBNs.

A **primary key** is a special case of unique keys. The major difference is that for unique keys the implicit NOT NULL constraint is not automatically enforced, while for primary keys it is enforced. Thus, the values in unique key columns may or may not be NULL. Another difference is that primary keys must be defined using another syntax.

Primary keys are defined with the following syntax:

```
CREATE TABLE table_name
(
    id_col INT,
    col2 CHARACTER VARYING(20),
    ...
    CONSTRAINT tab_pk PRIMARY KEY(id_col),
    ...
)
```

If the primary key consists only of a single column, the column can be marked as such using the following syntax:

```
CREATE TABLE table_name
(
    id_col INT PRIMARY KEY,
    col2 CHARACTER VARYING(20),
    ...
)
```

The definition of unique keys is syntactically very similar to primary keys.

Likewise, unique keys can be defined as part of the CREATE TABLE SQL statement.

```
CREATE TABLE table_name
(
    id_col INT,
    col2 CHARACTER VARYING(20),
    key_col SMALLINT,
    ...
    CONSTRAINT key_unique UNIQUE(key_col),
    ...
)
```

Or if the unique key consists only of a single column, the column can be marked as such using the following syntax:

```
CREATE TABLE table_name
(
```

```
id_col INT PRIMARY KEY,  
col2 CHARACTER VARYING(20),  
...  
key_col SMALLINT UNIQUE,  
...  
)
```

## 2.3 Foreign Key

In the context of relational databases, a foreign key is a referential constraint between two tables. The foreign key identifies a column or a set of columns in one table that refers to a column or set of columns in another table. The columns in the referencing table must be the primary key or other candidate key in the referenced table. The values in one row of the referencing columns must occur in a single row in the referenced table. Thus, a row in the referencing table cannot contain values that don't exist in the referenced table. This way references can be made to link information together and it is an essential part of database normalization. Multiple rows in the referencing table may refer to the same row in the referenced table. Most of the time, it reflects the one (master table, or referenced table) to many (child table, or referencing table) relationship.

The referencing and referenced table may be the same table, i.e. the foreign key refers back to the same table. Such a foreign key is known as self-referencing or recursive foreign key.

A table may have multiple foreign keys, and each foreign key can have a different referenced table. Each foreign key is enforced independently by the database system. Therefore, cascading relationships between tables can be established using foreign keys.

Improper foreign key/primary key relationships or not enforcing those relationships are often the source of many database and data modeling problems.

Foreign keys can be defined as part of the CREATE TABLE SQL statement.

```
CREATE TABLE table_name  
(  
    id INTEGER PRIMARY KEY,  
    col2 CHARACTER VARYING(20),  
    col3 INTEGER,  
    ...  
    CONSTRAINT col3_fk FOREIGN KEY(col3)  
        REFERENCES other_table(key_col),  
    ...  
)
```

If the foreign key is a single column only, the column can be marked as such using the following syntax:

```
CREATE TABLE table_name  
(  
    id INTEGER PRIMARY KEY,  
    col2 CHARACTER VARYING(20),  
    col3 INTEGER REFERENCES other_table(column_name),  
    ...  
)
```

```
...  
)
```

## 2.4 Views

In database theory, a view consists of a stored query accessible as a virtual table composed of the result set of a query. Unlike ordinary tables in a relational database, a view does not form part of the physical schema: it is a dynamic, virtual table computed or collated from data in the database. Changing the data in a table alters the data shown in subsequent invocations of the view.

Views can provide advantages over tables:

- Views can represent a subset of the data contained in a table
- Views can join and simplify multiple tables into a single virtual table
- Views can act as aggregated tables, where the database engine aggregates data (sum, average etc) and presents the calculated results as part of the data
- Views can hide the complexity of data; for example a view could appear as Sales2000 or Sales2001, transparently partitioning the actual underlying table
- Views take very little space to store; the database contains only the definition of a view, not a copy of all the data it presents
- Views can limit the degree of exposure of a table or tables to the outer world

Syntax:

```
CREATE VIEW <ViewName>  
AS  
...
```

## 2.5 Functions

In SQL databases, a user-defined function provides a mechanism for extending the functionality of the database server by adding a function that can be evaluated in SQL statements. The SQL standard distinguishes between scalar and table functions. A scalar function returns only a single value (or NULL), whereas a table function returns a (relational) table comprising zero or more rows, each row with one or more columns.

User-defined functions in SQL are declared using the CREATE FUNCTION statement.

Syntax:

```
CREATE FUNCTION <FunctionName>  
  (@Parameter1 <datatype>,  
   @ Parameter2 <datatype>,  
   ...)
```

```
RETURNS <datatype>  
AS  
...
```

## 2.6 Stored Procedures

A stored procedure is executable code that is associated with, and generally stored in, the database. Stored procedures usually collect and customize common operations, like inserting a tuple into a relation, gathering statistical information about usage patterns, or encapsulating complex business logic and calculations. Frequently they are used as an application programming interface (API) for security or simplicity.

Stored procedures are not part of the relational database model, but all commercial implementations include them.

Stored procedures are called or used with the following syntax:

```
CALL procedure (...)
```

or

```
EXECUTE procedure (...)
```

Stored procedures can return result sets, i.e. the results of a SELECT statement. Such result sets can be processed using cursors by other stored procedures by associating a result set locator, or by applications. Stored procedures may also contain declared variables for processing data and cursors that allow it to loop through multiple rows in a table. The standard Structured Query Language provides IF, WHILE, LOOP, REPEAT, CASE statements, and more. Stored procedures can receive variables, return results or modify variables and return them, depending on how and where the variable is declared.

## 2.7 Triggers

A database trigger is procedural code that is automatically executed in response to certain events on a particular table or view in a database. The trigger is mostly used for keeping the integrity of the information on the database. For example, when a new record (representing a new worker) added to the employees table, new records should be created also in the tables of the taxes, vacations, and salaries.

The syntax is as follows:

```
CREATE TRIGGER <TriggerName> ON <TableName>  
FOR INSERT, UPDATE, DELETE  
AS  
...
```

# 3 Structured Query Language (SQL)

SQL (Structured Query Language) is a database computer language designed for managing data in relational database management systems (RDBMS).

This document gives only a very brief overview of SQL, for more in-depth overview of SQL. Please refer to the tutorial “Structured Query Language” located on my web site:

<https://www.halvorsen.blog>

## 3.1 Queries

The most common operation in SQL is the query, which is performed with the declarative SELECT statement. SELECT retrieves data from one or more tables, or expressions. Standard SELECT statements have no persistent effects on the database.

Queries allow the user to describe desired data, leaving the database management system (DBMS) responsible for planning, optimizing, and performing the physical operations necessary to produce that result as it chooses.

A query includes a list of columns to be included in the final result immediately following the SELECT keyword. An asterisk ("\*") can also be used to specify that the query should return all columns of the queried tables. SELECT is the most complex statement in SQL, with optional keywords and clauses that include:

- The **FROM** clause which indicates the table(s) from which data is to be retrieved. The FROM clause can include optional JOIN subclauses to specify the rules for joining tables.
- The **WHERE** clause includes a comparison predicate, which restricts the rows returned by the query. The WHERE clause eliminates all rows from the result set for which the comparison predicate does not evaluate to True.
- The **GROUP BY** clause is used to project rows having common values into a smaller set of rows. GROUP BY is often used in conjunction with SQL aggregation functions or



to eliminate duplicate rows from a result set. The WHERE clause is applied before the GROUP BY clause.

- The **HAVING** clause includes a predicate used to filter rows resulting from the GROUP BY clause. Because it acts on the results of the GROUP BY clause, aggregation functions can be used in the HAVING clause predicate.
- The **ORDER BY** clause identifies which columns are used to sort the resulting data, and in which direction they should be sorted (options are ascending or descending). Without an ORDER BY clause, the order of rows returned by an SQL query is undefined.

### Example:

The following is an example of a SELECT query that returns a list of expensive books. The query retrieves all rows from the Book table in which the price column contains a value greater than 100.00. The result is sorted in ascending order by title. The asterisk (\*) in the select list indicates that all columns of the Book table should be included in the result set.

```
SELECT *  
FROM Book  
WHERE price > 100.00  
ORDER BY title;
```

The example below demonstrates a query of multiple tables, grouping, and aggregation, by returning a list of books and the number of authors associated with each book.

```
SELECT Book.title, count(*) AS Authors  
FROM Book  
JOIN Book_author ON Book.isbn = Book_author.isbn  
GROUP BY Book.title
```

[End of Example]

## 3.2 Data Manipulation

The **Data Manipulation Language (DML)** is the subset of SQL used to add, update and delete data.

The acronym **CRUD** refers to all of the major functions that need to be implemented in a relational database application to consider it complete. Each letter in the acronym can be mapped to a standard SQL statement:

Operation	SQL
Create	INSERT

Read (Retrieve)	SELECT
Update	UPDATE
Delete (Destroy)	DELETE

### Example:

INSERT:

INSERT adds rows to an existing table, e.g.,:

```
INSERT INTO My_table field1, field2, field3)
VALUES ('test', 'N', NULL)
```

UPDATE:

UPDATE modifies a set of existing table rows, e.g.,:

```
UPDATE My_table
SET field1 = 'updated value'
WHERE field2 = 'N'
```

DELETE:

DELETE removes existing rows from a table, e.g.,:

```
DELETE FROM My_table
WHERE field2 = 'N'
```

[End of Example]

## 3.3 Data Definition

The **Data Definition Language (DDL)** manages table and index structure. The most basic items of DDL are the CREATE, ALTER, RENAME and DROP statements:

- **CREATE** creates an object (a table, for example) in the database.
- **DROP** deletes an object in the database, usually irretrievably.
- **ALTER** modifies the structure an existing object in various ways—for example, adding a column to an existing table.

### Example:

CREATE:

Create a Database Table:

```
CREATE TABLE My_table
(
    my_field1 INT,
```

```
my_field2  VARCHAR(50),  
my_field3  DATE          NOT NULL,  
PRIMARY KEY (my_field1)  
)
```

[End of Example]

## 3.4 Data Types

Each column in an SQL table declares the type(s) that column may contain. ANSI SQL includes the following data types.

### 3.4.1 Character Strings

- CHARACTER(n) or CHAR(n) — fixed-width n-character string, padded with spaces as needed
- CHARACTER VARYING(n) or VARCHAR(n) — variable-width string with a maximum size of n characters
- NATIONAL CHARACTER(n) or NCHAR(n) — fixed width string supporting an international character set
- NATIONAL CHARACTER VARYING(n) or NVARCHAR(n) — variable-width NCHAR string

### 3.4.2 Bit Strings

- BIT(n) — an array of n bits
- BIT VARYING(n) — an array of up to n bits

### 3.4.3 Numbers

- INTEGER and SMALLINT
- FLOAT, REAL and DOUBLE PRECISION
- NUMERIC(precision, scale) or DECIMAL(precision, scale)

### 3.4.4 Date and Time

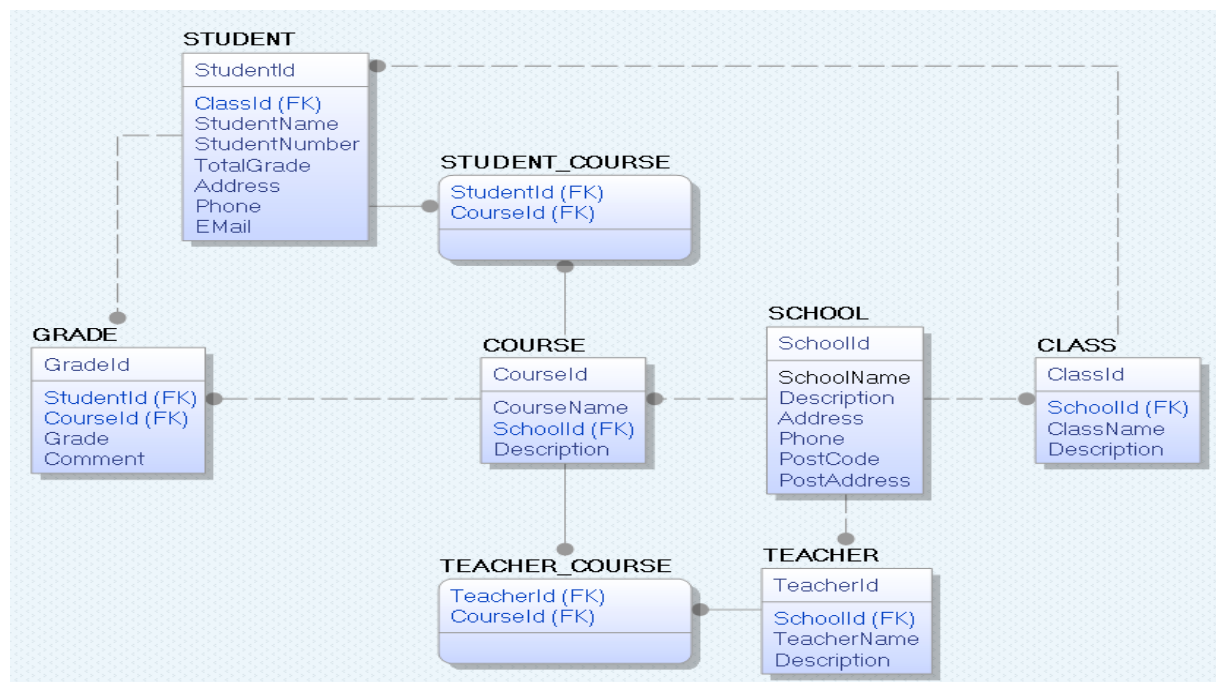
- DATE
- TIME
- TIMESTAMP
- INTERVAL

# 4 Database Modelling

## 4.1 ER Diagram

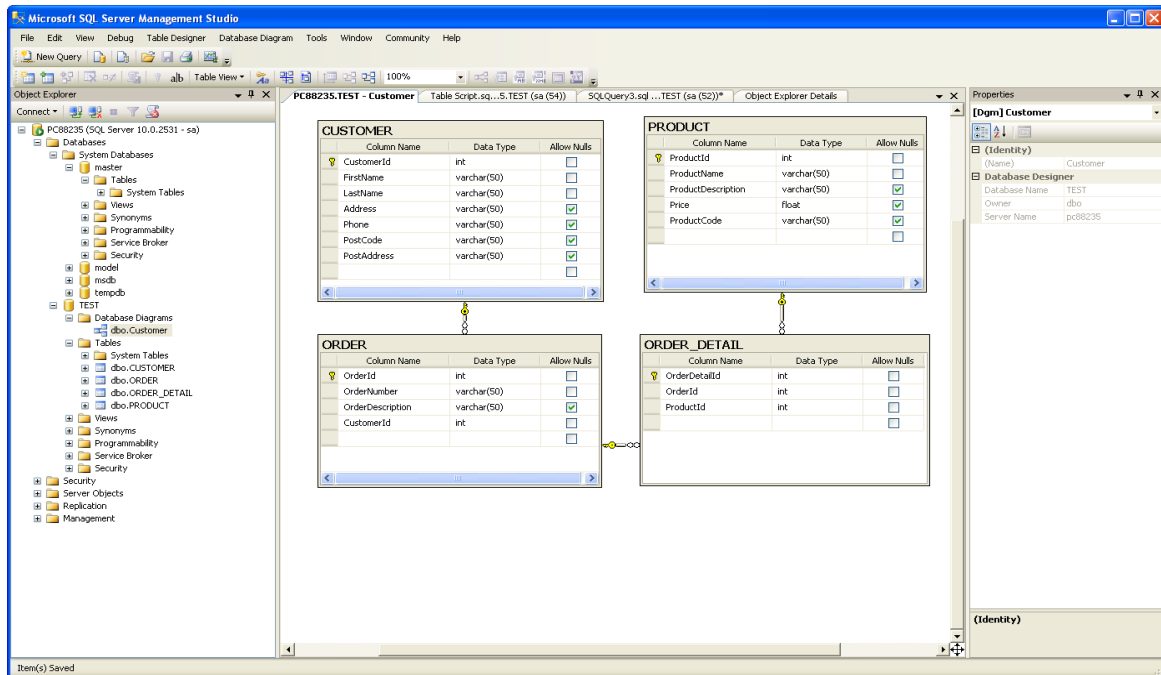
In software engineering, an Entity-Relationship Model (ERM) is an abstract and conceptual representation of data. Entity-relationship modeling is a database modeling method, used to produce a type of conceptual schema or semantic data model of a system, often a relational database, and its requirements in a top-down fashion.

Diagrams created using this process are called entity-relationship diagrams, or ER diagrams or ERDs for short.



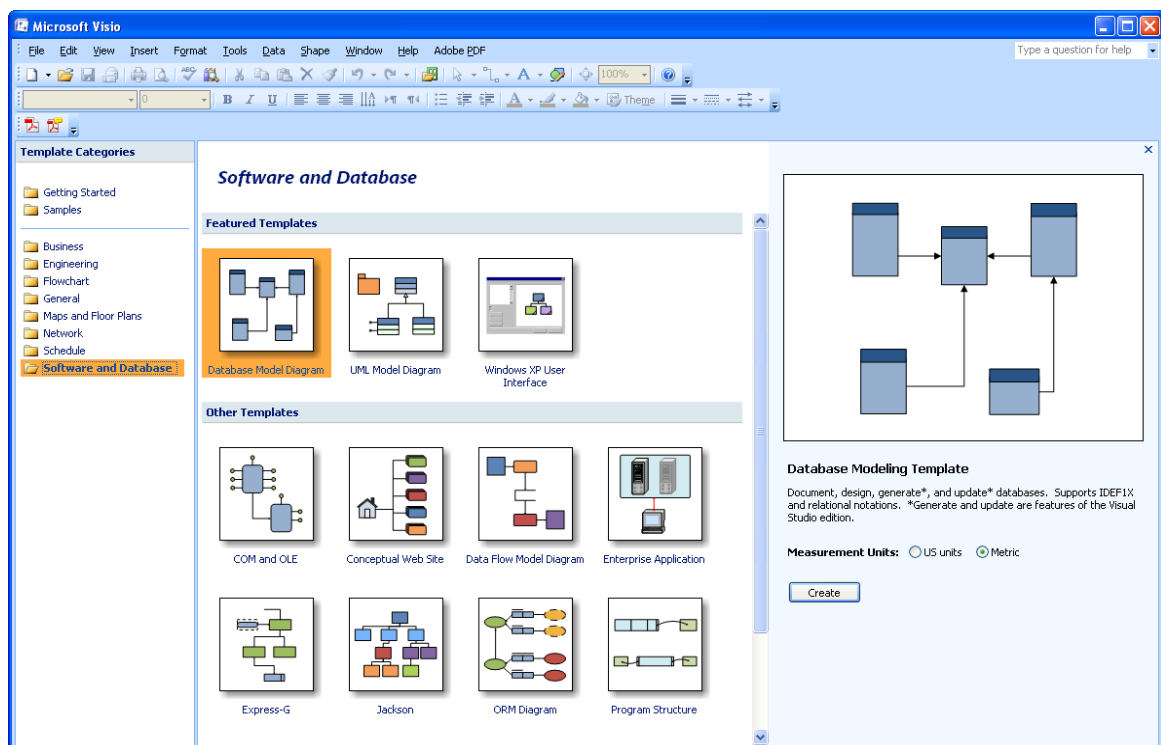
There are many ER diagramming tools. Some of the proprietary ER diagramming tools are ERwin, Enterprise Architect and Microsoft Visio.

Microsoft SQL Server has also a built-in tool for creating Database Diagrams.

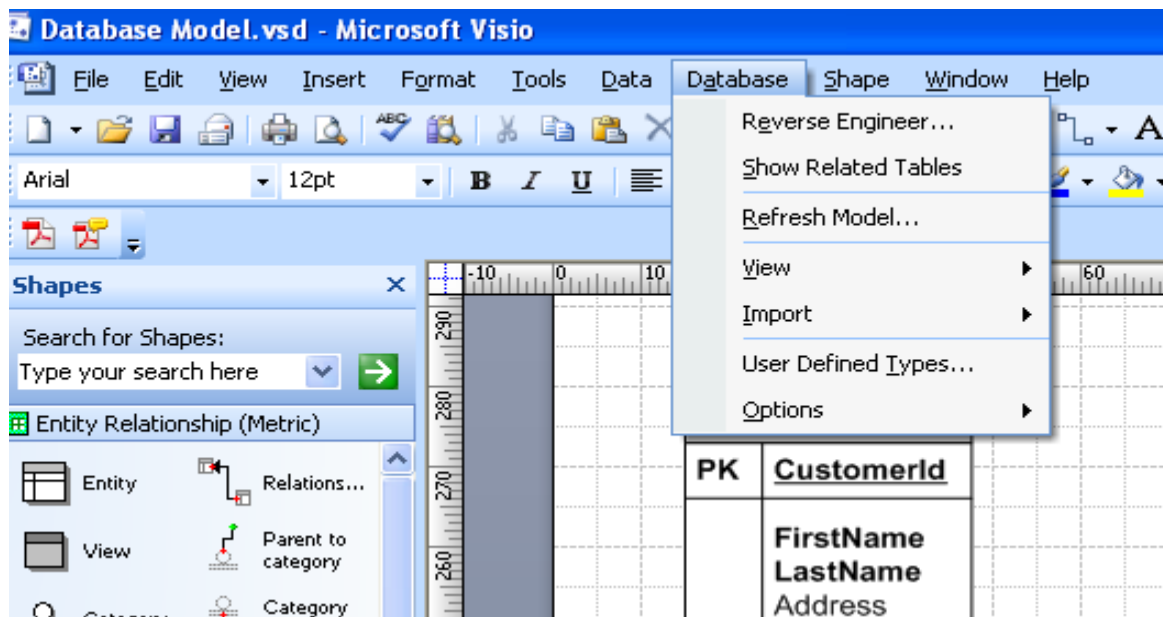


## 4.2 Microsoft Visio

Microsoft Visio is a diagramming program for creating different kinds of diagrams. Visio have a template for creating Database Model Diagrams.



In the Database menu Visio offers lots of functionality regarding your database model.



“Reverse Engineering” is the opposite procedure, i.e., extraction of a database schema from an existing database into a database model in Microsoft Visio.

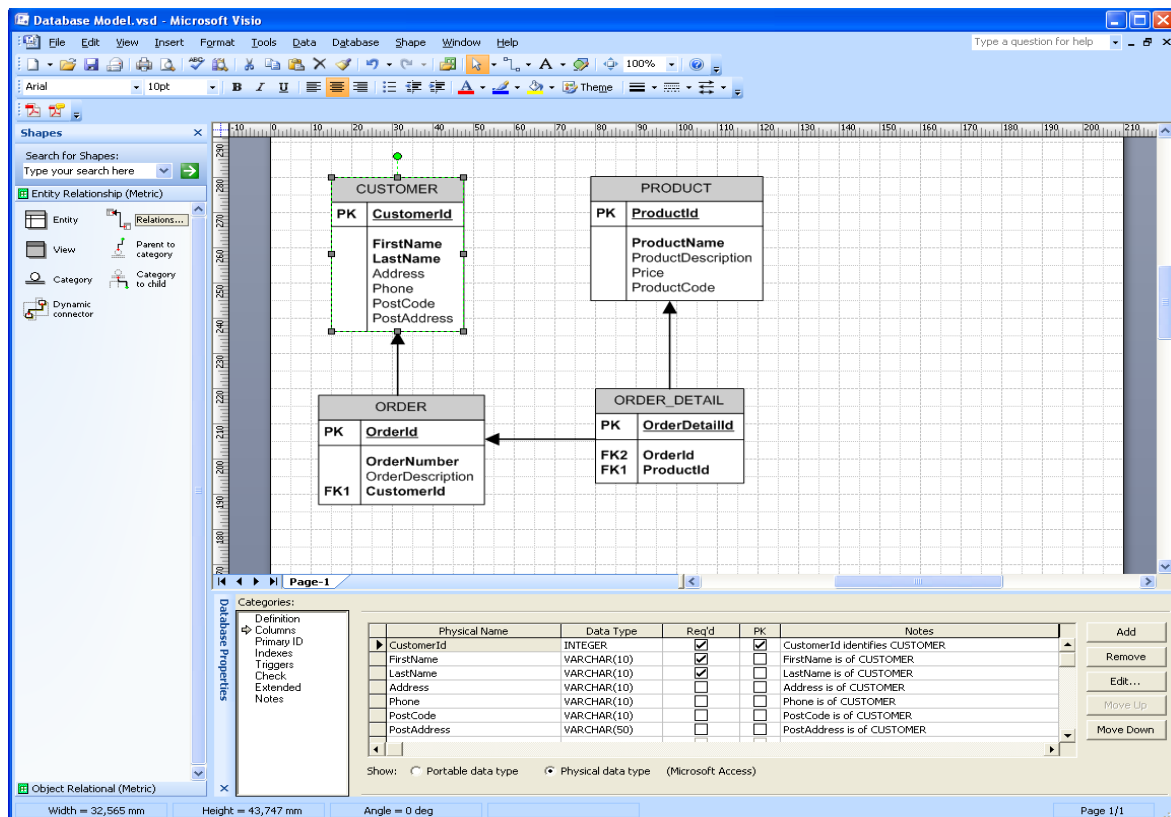
### Example: Database Diagram

Create the following tables in an ER Diagram using MS Visio.

- CUSTOMER
  - **CustomerId (PK)**
  - FirstName
  - LastName
  - Address
  - Phone
  - PostCode
  - PostAddress
- PRODUCT
  - **ProductId (PK)**
  - ProductName
  - ProductDescription
  - Price
  - ProductCode
- ORDER
  - **OrderId (PK)**
  - OrderNumber
  - OrderDescription

- **CustomerId (FK)**
- **ORDER\_DETAIL**
  - OrderDetailId (PK)
  - **OrderId (FK)**
  - **ProductId (FK)**

The Database Diagram becomes:



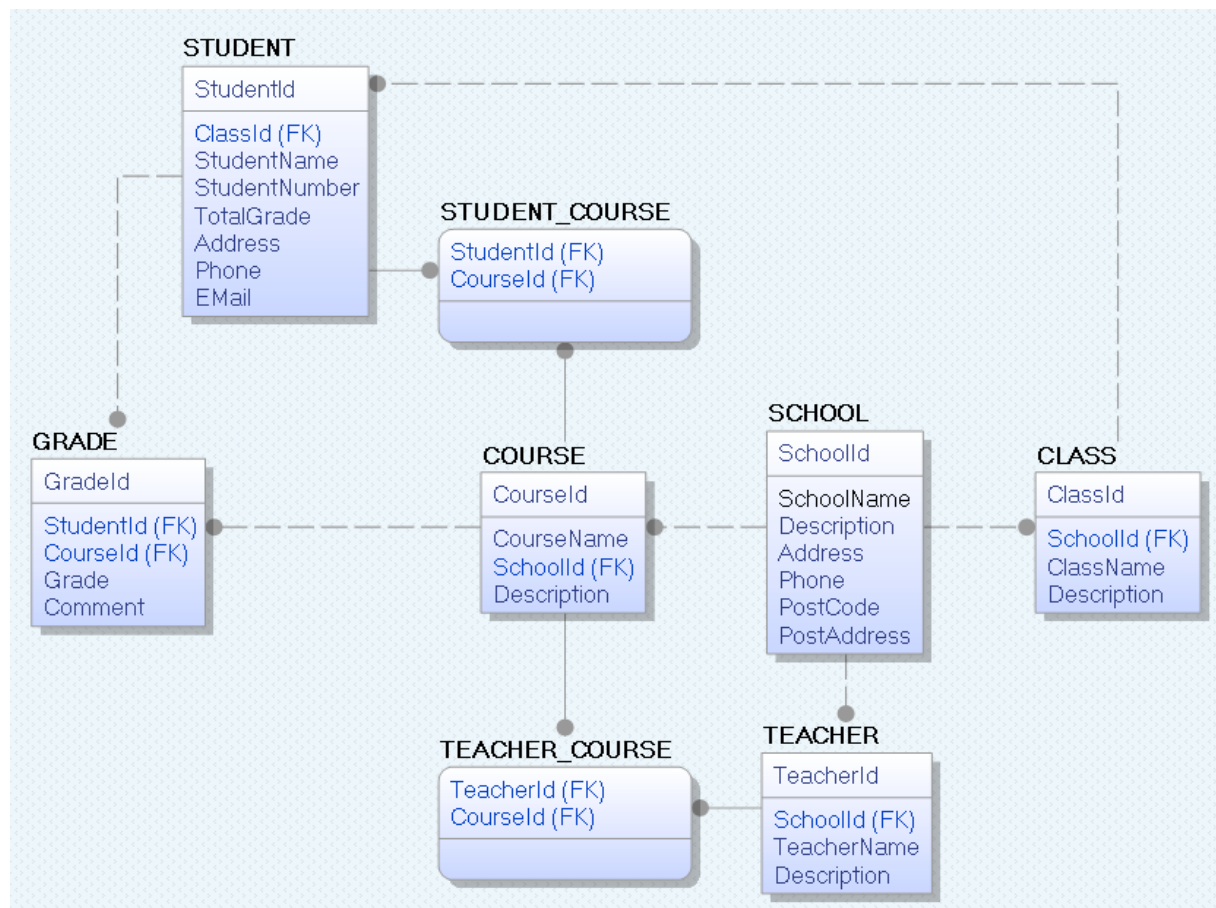
[End of Example]

## 4.3 ERwin

ERwin is a professional database modelling tool. A Community edition is also available for free. The Community edition is limited to work with max 25 objects.

Below we see an example created in Erwin.

With Erwin and other professional database modelling tools you can directly import the database model into the database system such as SQL Server, MySQL, etc.





# 5 Microsoft SQL Server

## 5.1 Introduction

Microsoft SQL Server is a relational model database server produced by Microsoft. Its primary query languages are T-SQL and ANSI SQL.

The latest version is Microsoft SQL Server 2014.

Microsoft SQL Server homepage: [www.microsoft.com/sqlserver](http://www.microsoft.com/sqlserver)

The Microsoft SQL Server comes in different versions, such as:

- SQL Server Developer Edition
- SQL Server Enterprise Edition
- SQL Server Web Edition
- SQL Server Express Edition
- Etc.

The SQL Server Express Edition is a freely-downloadable and -distributable version.

## 5.2 SQL Server Express

The SQL Server Express Edition is a freely-downloadable and -distributable version.

However, the Express edition has a number of technical restrictions which make it undesirable for large-scale deployments, including:

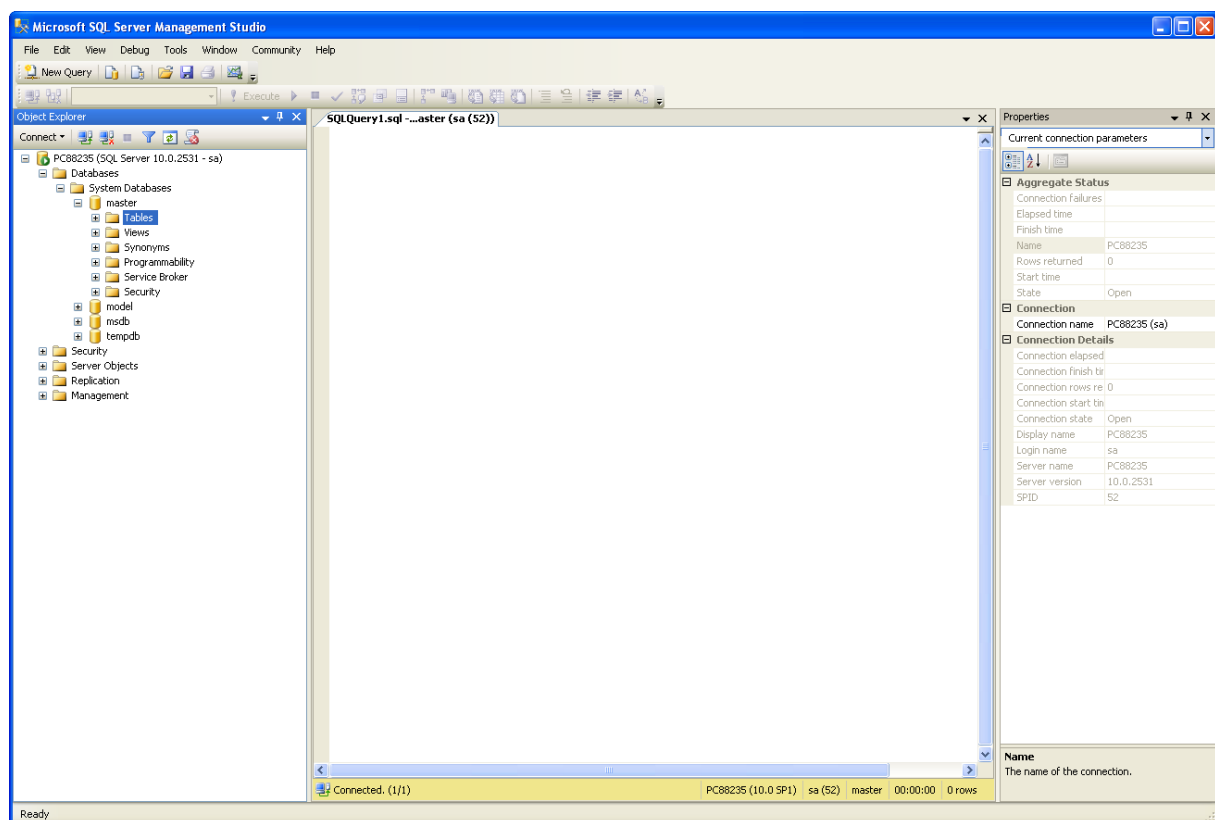
- Maximum database size of 4 GB per. The 4 GB limit applies per database (log files excluded); but in some scenarios users can access more data through the use of multiple interconnected databases.
- Single physical CPU, multiple cores
- 1 GB of RAM (runs on any size RAM system, but uses only 1 GB)

SQL Server Express offers a GUI tools for database management in a separate download and installation package, called **SQL Server Management Studio Express**.

## 5.3 SQL Server Management Studio

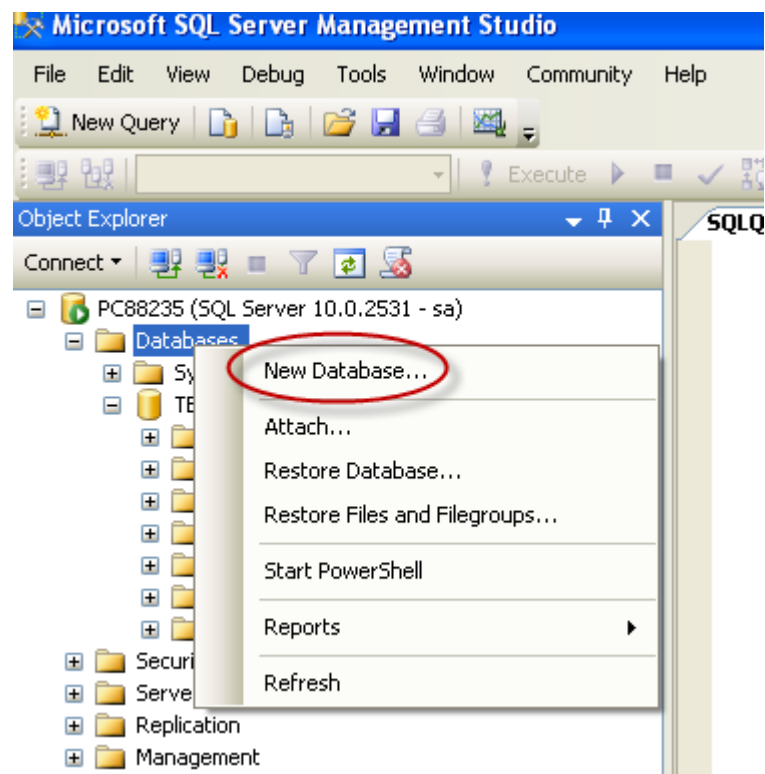
SQL Server Management Studio is a GUI tool included with SQL Server for configuring, managing, and administering all components within Microsoft SQL Server. The tool includes both script editors and graphical tools that work with objects and features of the server. As mentioned earlier, version of SQL Server Management Studio is also available for SQL Server Express Edition, for which it is known as SQL Server Management Studio Express.

A central feature of SQL Server Management Studio is the Object Explorer, which allows the user to browse, select, and act upon any of the objects within the server. It can be used to visually observe and analyze query plans and optimize the database performance, among others. SQL Server Management Studio can also be used to create a new database, alter any existing database schema by adding or modifying tables and indexes, or analyze performance. It includes the query windows which provide a GUI based interface to write and execute queries.

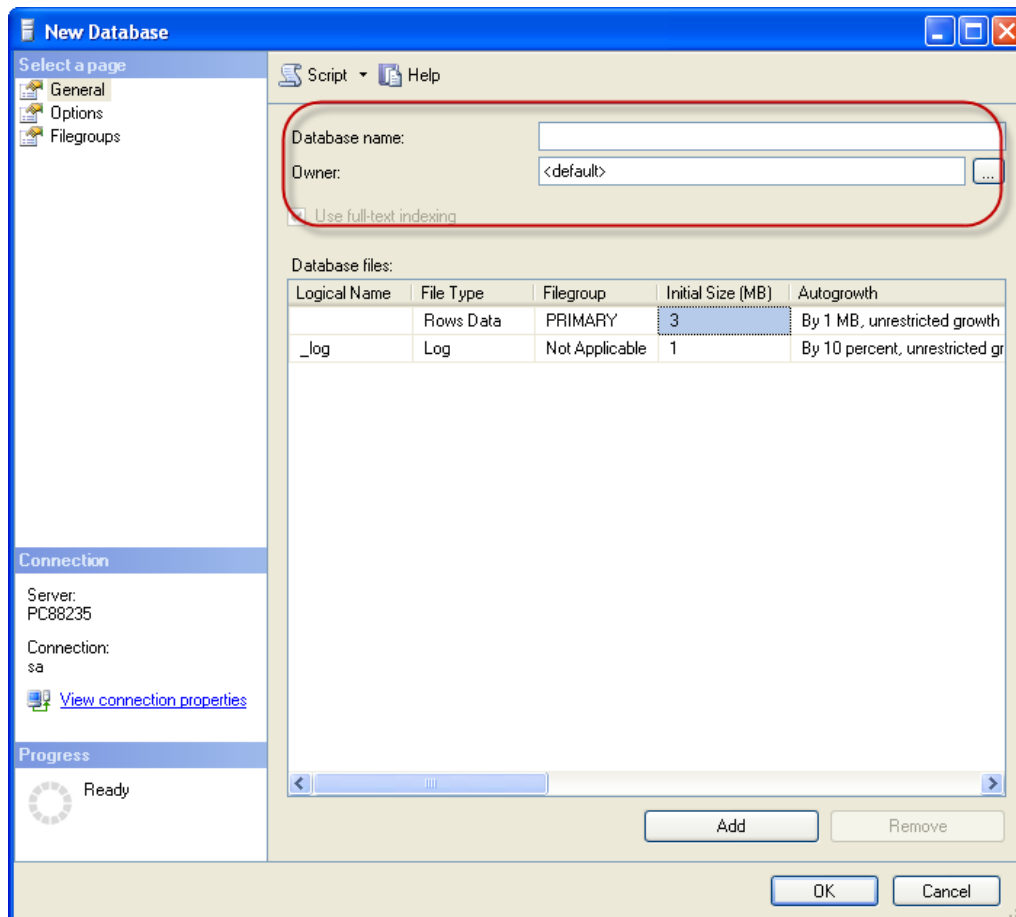


## 5.4 Create a New Database

It is quite simple to create a new database in Microsoft SQL Server. Just right-click on the “Databases” node and select “New Database...”

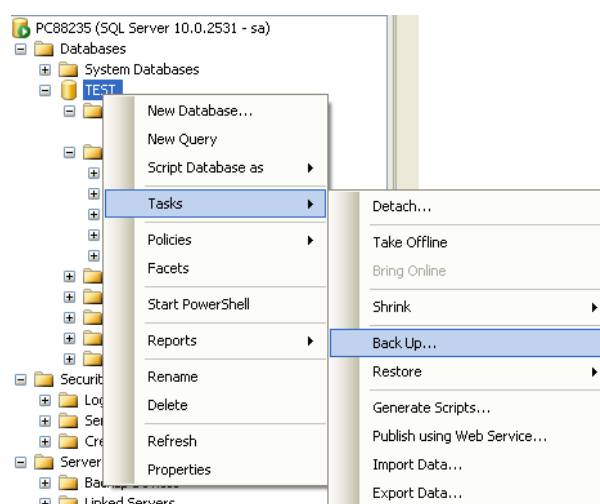


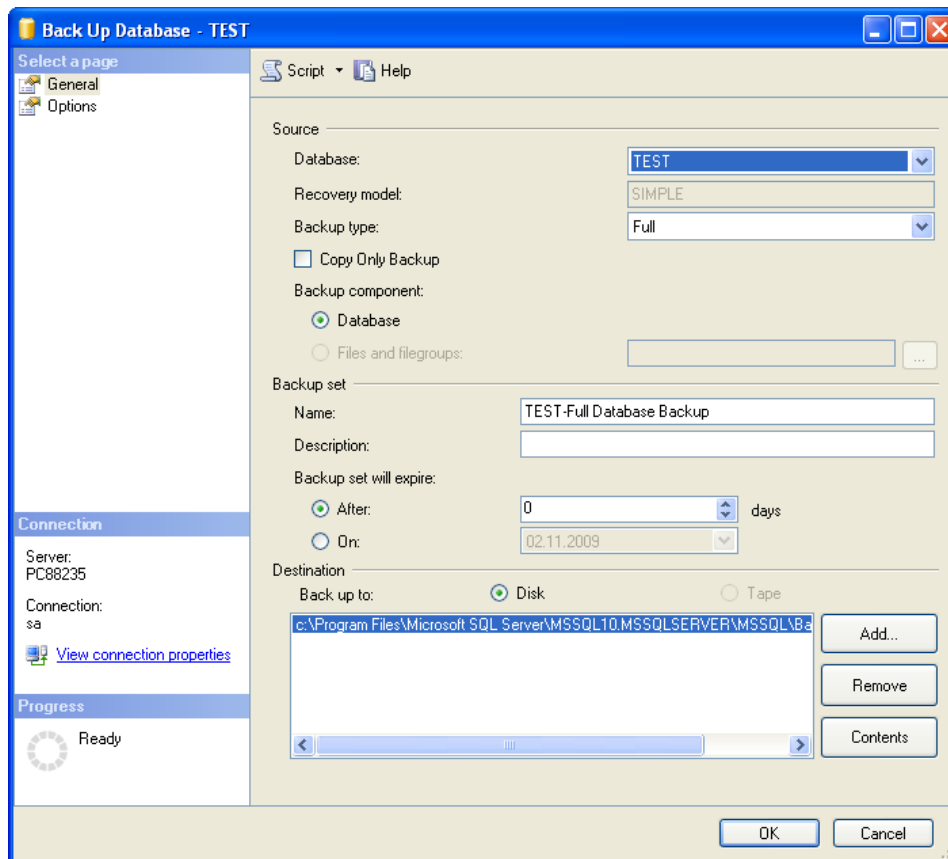
There are lots of settings you may set regarding your database, but the only information you must fill in is the name of your database:



## 5.5 Backup/Restore

Database Backup and Restore:





# 6 Microsoft Office Access

## 6.1 Introduction

Microsoft Office Access, previously known as Microsoft Access, is a relational database management system from Microsoft that combines the relational Microsoft Jet Database Engine with a graphical user interface and software development tools. It is a member of the Microsoft Office suite of applications and is included in the Professional and higher versions for Windows. Access stores data in its own format based on the Access Jet Database Engine.

Microsoft Access is used by programmers and non-programmers to create their own simple database solutions.

Microsoft Access is a file server-based database. Unlike client-server relational database management systems (RDBMS), e.g., Microsoft SQL Server, Microsoft Access does not implement database triggers, stored procedures, or transaction logging. All database tables, queries, forms, reports, macros, and modules are stored in the Access Jet database as a single file. This makes Microsoft Access useful in small applications, teaching, etc. because it is easy to move from one computer to another.

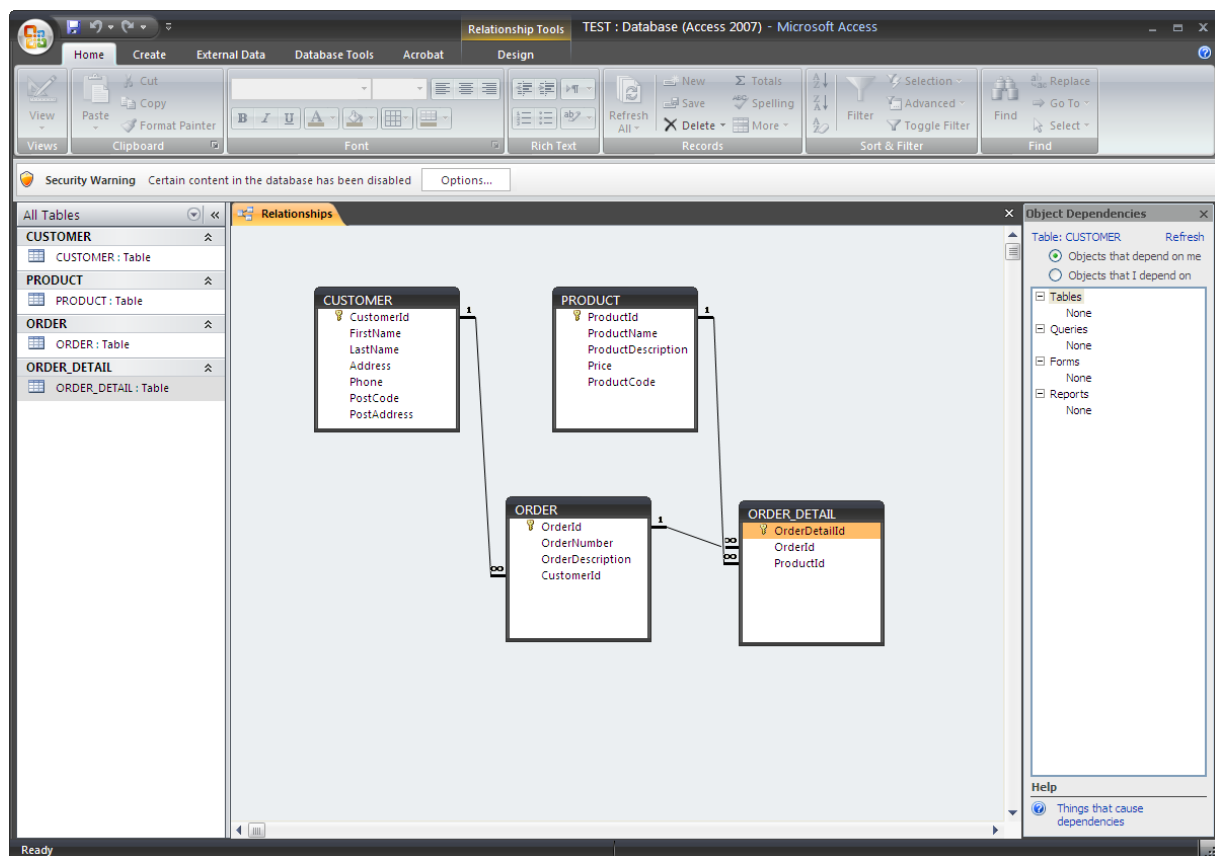
## 6.2 Example Database

I will present an example database in Microsoft Access 2007 which will be used in some of the examples and exercises in this document.

The database consists of the following tables:

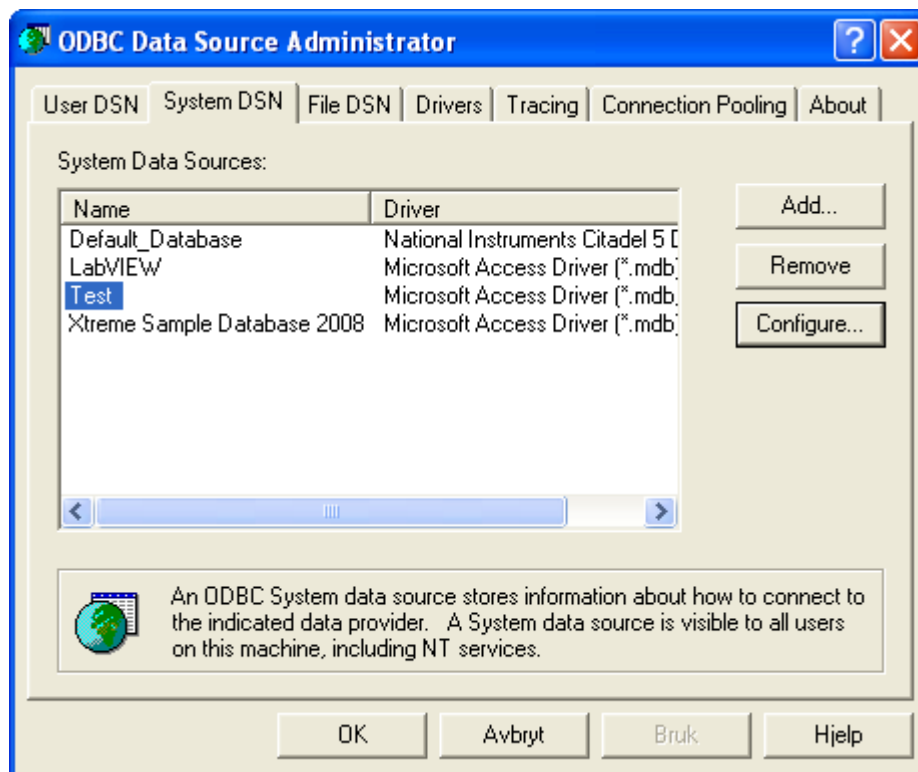
- CUSTOMER
  - **CustomerId (PK)**
  - FirstName
  - LastName
  - Address
  - Phone
  - PostCode
  - PostAddress
- PRODUCT

- **ProductId (PK)**
- ProductName
- ProductDescription
- Price
- ProductCode
- **ORDER**
  - **OrderId (PK)**
  - OrderNumber
  - OrderDescription
  - **CustomerId (FK)**
- **ORDER\_DETAIL**
  - OrderDetailId (PK)
  - **OrderId (FK)**
  - **ProductId (FK)**



ODBC Connection:

Administrative Tools → Data Sources (ODBC)





# 7 Creating and Using Tables

The SQL syntax for creating a Table is as follows:

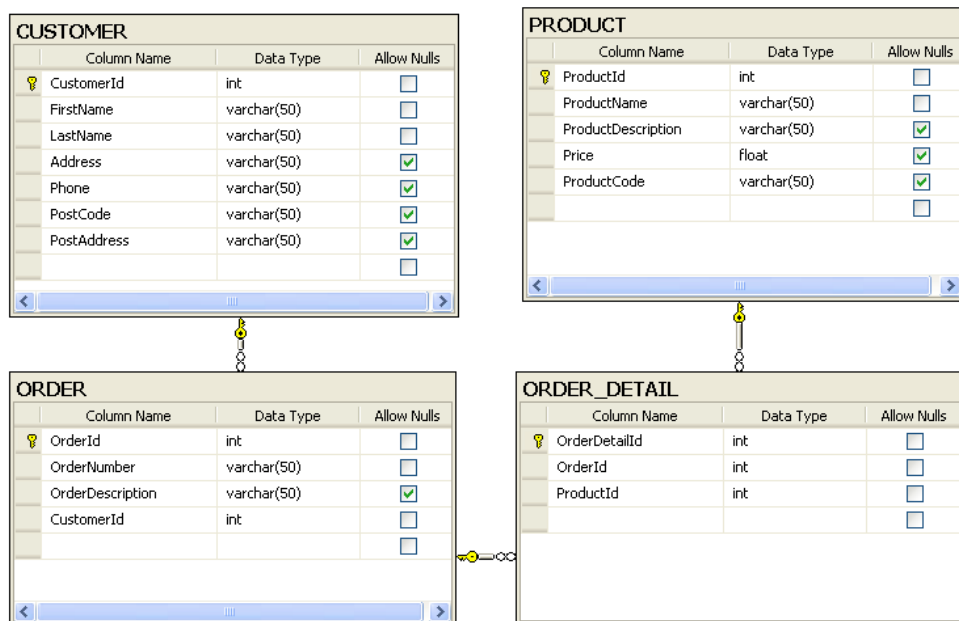
```
CREATE TABLE <TableName>
(
  <ColumnName> <datatype>
  ...
)
```

The SQL syntax for inserting Data into a Table is as follows:

```
INSERT INTO <TableName> (<Column1>, <Column2>, ...)
VALUES (<Data for Column1>, <Data for Column2>, ...)
```

## Example: Insert Data into Tables

We will insert some data into our tables:



The following SQL Query inserts some example data into these tables:

```
--CUSTOMER
INSERT INTO [CUSTOMER] ([FirstName],[LastName],[Address],[Phone],[PostCode],[PostAddress])
VALUES ('Per', 'Nilsen', 'Vipeveien 12', '12345678', '1234', 'Porsgrunn')
GO
```

```

INSERT INTO [CUSTOMER] ([FirstName],[LastName],[Address],[Phone],[PostCode],[PostAddress])
VALUES ('Tor', 'Hansen', 'Vipeveien 15', '77775678', '4455', 'Bergen')
GO
INSERT INTO [CUSTOMER] ([FirstName],[LastName],[Address],[Phone],[PostCode],[PostAddress])
VALUES ('Arne', 'Nilsen', 'Vipeveien 17', '12345778', '4434', 'Porsgrunn')
GO

--PRODUCT
INSERT INTO [PRODUCT] ([ProductName],[ProductDescription],[Price],[ProductCode]) VALUES
('Product A', 'This is product A', 1000, 'A-1234')
GO
INSERT INTO [PRODUCT] ([ProductName],[ProductDescription],[Price],[ProductCode]) VALUES
('Product B', 'This is product B', 1000, 'B-1234')
GO
INSERT INTO [PRODUCT] ([ProductName],[ProductDescription],[Price],[ProductCode]) VALUES
('Product C', 'This is product C', 1000, 'C-1234')
GO

--ORDER
INSERT INTO [ORDER] ([OrderNumber],[OrderDescription],[CustomerId]) VALUES ('10001', 'This is
Order 10001', 1)
GO
INSERT INTO [ORDER] ([OrderNumber],[OrderDescription],[CustomerId]) VALUES ('10002', 'This is
Order 10002', 2)
GO
INSERT INTO [ORDER] ([OrderNumber],[OrderDescription],[CustomerId]) VALUES ('10003', 'This is
Order 10003', 3)
GO

--ORDER_DETAIL
INSERT INTO [ORDER_DETAIL] ([OrderId],[ProductId]) VALUES (1, 1)
GO
INSERT INTO [ORDER_DETAIL] ([OrderId],[ProductId]) VALUES (1, 2)
GO
INSERT INTO [ORDER_DETAIL] ([OrderId],[ProductId]) VALUES (1, 3)
GO
INSERT INTO [ORDER_DETAIL] ([OrderId],[ProductId]) VALUES (2, 1)
GO
INSERT INTO [ORDER_DETAIL] ([OrderId],[ProductId]) VALUES (2, 2)
GO
INSERT INTO [ORDER_DETAIL] ([OrderId],[ProductId]) VALUES (3, 3)
GO
INSERT INTO [ORDER_DETAIL] ([OrderId],[ProductId]) VALUES (3, 1)
GO
INSERT INTO [ORDER_DETAIL] ([OrderId],[ProductId]) VALUES (3, 2)
GO
INSERT INTO [ORDER_DETAIL] ([OrderId],[ProductId]) VALUES (3, 3)
GO

```

Executing the following Queries then gives:

```
select * from CUSTOMER
```

	CustomerId	FirstName	LastName	Address	Phone	PostCode	PostAddress
1	1	Per	Nilsen	Vipeveien 12	12345678	1234	Porsgrunn
2	2	Tor	Hansen	Vipeveien 15	77775678	4455	Bergen
3	3	Arne	Nilsen	Vipeveien 17	12345778	4434	Porsgrunn

```
select * from PRODUCT
```

Results		Messages			
	ProductId	ProductName	ProductDescription	Price	ProductCode
1	1	Product A	This is product A	1000	A-1234
2	2	Product B	This is product B	1000	B-1234
3	3	Product C	This is product C	1000	C-1234

```
select * from [ORDER]
```

Results		Messages		
	OrderId	OrderNumber	OrderDescription	CustomerId
1	1	10001	This is Order 10001	1
2	2	10002	This is Order 10002	2
3	3	10003	This is Order 10003	3

```
select * from ORDER_DETAIL
```

Results		Messages	
	OrderDetailId	OrderId	ProductId
1	1	1	1
2	2	1	2
3	3	1	3
4	4	2	1
5	5	2	2
6	6	3	3
7	7	3	1
8	8	3	2
9	9	3	3

[End of Example]

# 8 Creating and Using Views

In database theory, a view consists of a stored query accessible as a virtual table composed of the result set of a query. Unlike ordinary tables in a relational database, a view does not form part of the physical schema: it is a dynamic, virtual table computed or collated from data in the database. Changing the data in a table alters the data shown in subsequent invocations of the view.

Views can provide advantages over tables:

- Views can represent a subset of the data contained in a table
- Views can join and simplify multiple tables into a single virtual table
- Views can act as aggregated tables, where the database engine aggregates data (sum, average etc) and presents the calculated results as part of the data
- Views can hide the complexity of data; for example a view could appear as Sales2000 or Sales2001, transparently partitioning the actual underlying table
- Views take very little space to store; the database contains only the definition of a view, not a copy of all the data it presents
- Depending on the SQL engine used, views can provide extra security
- Views can limit the degree of exposure of a table or tables to the outer world

Just as functions (in programming) can provide abstraction, so database users can create abstraction by using views. In another parallel with functions, database users can manipulate nested views, thus one view can aggregate data from other views.

Syntax:

```
CREATE VIEW <ViewName>  
AS  
...
```

# 9 Creating and using Stored Procedures

A stored procedure is a subroutine available to applications accessing a relational database system. Typical uses for stored procedures include data validation (integrated into the database) or access control mechanisms. Furthermore, stored procedures are used to consolidate and centralize logic that was originally implemented in applications. Large or complex processing that might require the execution of several SQL statements is moved into stored procedures, and all applications call the procedures only.

A stored procedure is a precompiled collection of SQL statements and optional control-of-flow statements, similar to a macro. Each database and data provider supports stored procedures differently. Stored procedures offer the following benefits to your database applications:

**Performance**—Stored Procedures are usually more efficient and faster than regular SQL queries because SQL statements are parsed for syntactical accuracy and precompiled by the DBMS when the stored procedure is created. Also, combining a large number of SQL statements with conditional logic and parameters into a stored procedure allows the procedures to perform queries, make decisions, and return results without extra trips to the database server.

**Maintainability**—Stored Procedures isolate the lower-level database structure from the application. As long as the table names, column names, parameter names, and types do not change from what is stated in the stored procedure, you do not need to modify the procedure when changes are made to the database schema. Stored procedures are also a way to support modular SQL programming because after you create a procedure, you and other users can reuse that procedure without knowing the details of the tables involved.

**Security**—When creating tables in a database, the Database Administrator can set EXECUTE permissions on stored procedures without granting SELECT, INSERT, UPDATE, and DELETE permissions to users. Therefore, the data in these tables is protected from users who are not using the stored procedures.

Stored procedures are similar to user-defined functions. The major difference is that functions can be used like any other expression within SQL statements, whereas stored procedures must be invoked using the CALL statement.

The syntax for creating a Stored Procedure is as follows:

```
CREATE PROCEDURE <ProcedureName>
@<Parameter1> <datatype>
...
```

### Example:

#### Create a Stored Procedure:

This Procedure gets Customer Data based on a specific Order Number.

```
IF EXISTS (SELECT name
            FROM    sysobjects
            WHERE   name = 'sp_CustomerOrders'
            AND     type = 'p')
    DROP PROCEDURE sp_CustomerOrders
GO

CREATE PROCEDURE sp_CustomerOrders
@OrderNumber varchar(50)
AS

/*-----
Last Updated Date:    2009.11.03
Last Updated By:      hans.pr.halvorsen@hit.no
Description:          Get Customer Information from a specific Order Number
-----*/
SET NOCOUNT ON

declare @CustomerId int

select @CustomerId = CustomerId from [ORDER] where OrderNumber = @OrderNumber

select CustomerId, FirstName, LastName, [Address], Phone from CUSTOMER where
CustomerId=@CustomerId

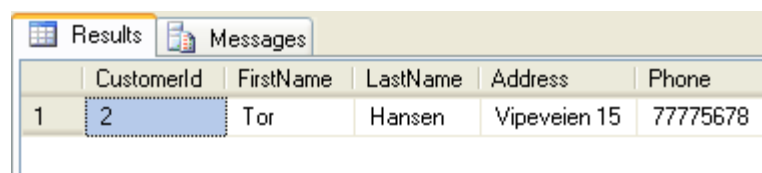
SET NOCOUNT OFF
Og
```

#### Using a Stored Procedure:

Using the Stored procedure like this

```
exec sp_CustomerOrders '10002'
```

This gives the following result:



Results		Messages			
	CustomerId	FirstName	LastName	Address	Phone
1	2	Tor	Hansen	Vipeveien 15	77775678

[End of Example]

# 10 Creating and Using Triggers

A database trigger is procedural code that is automatically executed in response to certain events on a particular table or view in a database. The trigger is mostly used for keeping the integrity of the information on the database. For example, when a new record (representing a new worker) added to the employees table, new records should be created also in the tables of the taxes, vacations, and salaries.

Triggers are commonly used to:

- prevent changes (e.g. prevent an invoice from being changed after it's been mailed out)
- log changes (e.g. keep a copy of the old data)
- audit changes (e.g. keep a log of the users and roles involved in changes)
- enhance changes (e.g. ensure that every change to a record is time-stamped by the server's clock, not the client's)
- enforce business rules (e.g. require that every invoice have at least one line item)
- execute business rules (e.g. notify a manager every time an employee's bank account number changes)
- replicate data (e.g. store a record of every change, to be shipped to another database later)
- enhance performance (e.g. update the account balance after every detail transaction, for faster queries)

The major features of database triggers, and their effects, are:

- do not accept parameters or arguments (but may store affected-data in temporary tables)
- cannot perform commit or rollback operations because they are part of the triggering SQL statement
- can cancel a requested operation
- can cause mutating table errors, if they are poorly written.

Microsoft SQL Server supports triggers either after or instead of an insert, update, or delete operation.

Syntax:

```
CREATE TRIGGER <TriggerName> ON <TableName>  
FOR INSERT, UPDATE, DELETE  
AS  
...
```



# 11 Creating and Using Functions

In SQL databases, a user-defined function provides a mechanism for extending the functionality of the database server by adding a function that can be evaluated in SQL statements. The SQL standard distinguishes between scalar and table functions. A scalar function returns only a single value (or NULL), whereas a table function returns a (relational) table comprising zero or more rows, each row with one or more columns.

Stored Procedures vs. Functions:

- Only functions can return a value (using the RETURN keyword).
- Stored procedures can use RETURN keyword but without any value being passed[1]
- Functions could be used in SELECT statements, provided they don't do any data manipulation and also should not have any OUT or IN OUT parameters.
- Functions must return a value, but for stored procedures this is not compulsory.
- A function can have only IN parameters, while stored procedures may have OUT or IN OUT parameters.
- A function is a subprogram written to perform certain computations and return a single value.
- A stored procedure is a subprogram written to perform a set of actions, and can return multiple values using the OUT parameter or return no value at all.

User-defined functions in SQL are declared using the CREATE FUNCTION statement.

Syntax:

```
CREATE FUNCTION <FunctionName>
    (@Parameter1 <datatype>,
    @ Parameter2 <datatype>,
    ...)
RETURNS <datatype>
AS
...
```



# Introduction to Database Systems

Hans-Petter Halvorsen

Copyright © 2017

E-Mail: [hans.p.halvorsen@usn.no](mailto:hans.p.halvorsen@usn.no)

Web: <https://www.halvorsen.blog>



<https://www.halvorsen.blog>