

Unit 1 - Chapter 1: Introduction to SQL

Overview of Database

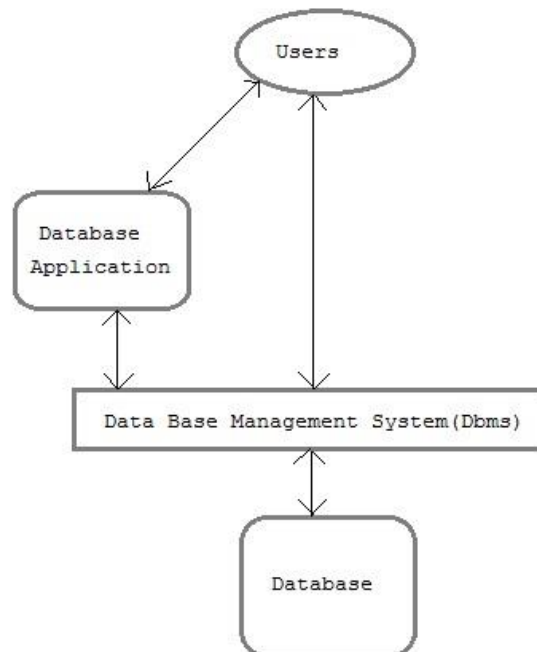
A **Database** is a collection of related data organized in a way that data can be easily accessed, managed and updated. Any piece of information can be a data, for example name of your school. Database is actually a place where related piece of information is stored and various operations can be performed on it.

DBMS

A **DBMS** is a software that allows creation, definition and manipulation of database. Dbms is actually a tool used to perform any kind of operation on data in database. Dbms also provides protection and security to database. It maintains data consistency in case of multiple users. Here are some examples of popular dbms, MySql, Oracle, Sybase, Microsoft Access and IBM DB2 etc.

Architecture of Database System

The database system can be divided into four components.



- **Users:** Users may be of various type such as DB administrator, System developer and End users.

- **Database application** : Database application may be Personal, Departmental, Enterprise and Internal
- **DBMS**: Software that allow users to define, create and manages database access, Ex: MySQL, Oracle etc.
- **Database**: Collection of logical data.

Functions of DBMS

- Provides data Independence
- Concurrency Control
- Provides Recovery services
- Provides Utility services
- Provides a clear and logical view of the process that manipulates data.

Advantages of DBMS

- Segregation of application program.
- Minimal data duplicacy.
- Easy retrieval of data.
- Reduced development time and maintenance need.

Disadvantages of DBMS

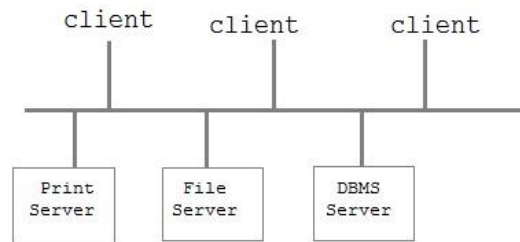
- Complexity
- Costly
- Large in size

Database Architecture

Database architecture is logically divided into two types.

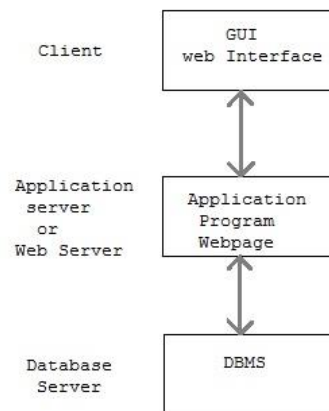
1. Logical two-tier Client / Server architecture
2. Logical three-tier Client / Server architecture

Two-tier Client / Server Architecture



Two-tier Client / Server architecture is used for User Interface program and Application Programs that runs on client side. An interface called ODBC(Open Database Connectivity) provides an API that allow client side program to call the dbms. Most DBMS vendors provide ODBC drivers. A client program may connect to several DBMS's. In this architecture some variation of client is also possible for example in some DBMS's more functionality is transferred to the client including data dictionary, optimization etc. Such clients are called **Data server**.

Three-tier Client / Server Architecture



Three-tier Client / Server database architecture is commonly used architecture for web applications. Intermediate layer called **Application server** or Web Server stores the web connectivity software and the business logic(constraints) part of application used to access the right amount of data from the database server. This layer acts like medium for sending partially processed data between the database server and the client.

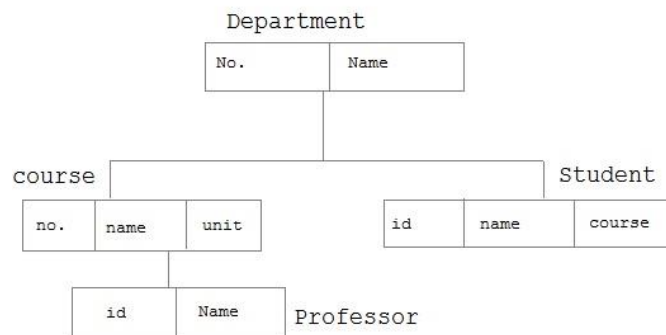
Database Model

A Database model defines the logical design of data. The model describes the relationships between different parts of the data. Historically, in database design, three models are commonly used. They are,

- Hierarchical Model
- Network Model
- Relational Model

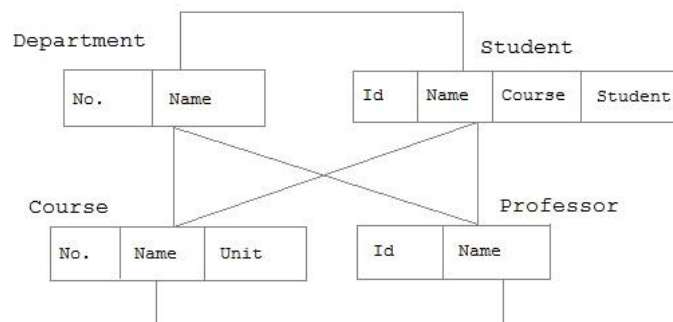
Hierarchical Model

In this model each entity has only one parent but can have several children. At the top of hierarchy there is only one entity which is called **Root**.



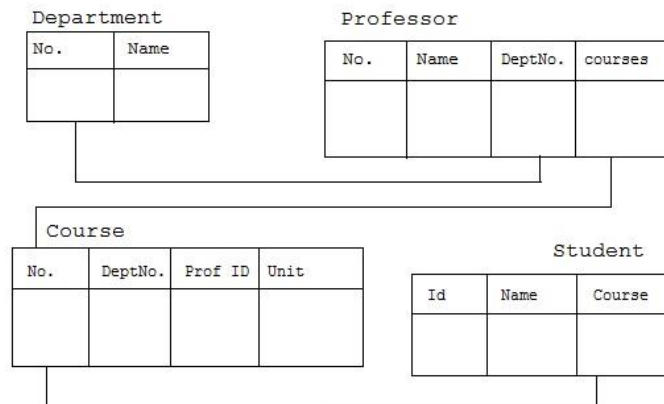
Network Model

In the network model, entities are organized in a graph, in which some entities can be accessed through several path



Relational Model

In this model, data is organized in two-dimensional tables called **relations**. The tables or relation are related to each other.



Codd's Rule

E.F Codd was a Computer Scientist who invented **Relational model** for Database management. Based on relational model, **Relation database** was created. Codd proposed 13 rules popularly known as **Codd's 12 rules** to test DBMS's concept against his relational model. Codd's rule actually define what quality a DBMS requires in order to become a Relational Database Management System(RDBMS). Till now, there is hardly any commercial product that follows all the 13 Codd's rules. Even **Oracle** follows only eight and half out(8.5) of 13. The Codd's 12 rules are as follows.

Rule zero

This rule states that for a system to qualify as an **RDBMS**, it must be able to manage database entirely through the relational capabilities.

Rule 1 : Information rule

All information(including metadata) is to be represented as stored data in cells of tables. The rows and columns have to be strictly unordered.

Rule 2 : Guaranteed Access

Each unique piece of data(atomic value) should be accessible by : **Table Name + primary key(Row) + Attribute(column)**.

NOTE : Ability to directly access via POINTER is a violation of this rule.

Rule 3 : Systematic treatment of NULL

Null has several meanings, it can mean missing data, not applicable or no value. It should be handled consistently. Primary key must not be null. Expression on **NULL** must give null.

Rule 4 : Active Online Catalog

Database dictionary(catalog) must have description of **Database**. Catalog to be governed by same rule as rest of the database. The same query language to be used on catalog as on application database.

Rule 5 : Powerful language

One well defined language must be there to provide all manners of access to data.

Example: **SQL**. If a file supporting table can be accessed by any manner except SQL interface, then its a violation to this rule.

Rule 6 : View Updation rule

All view that are theoretically updatable should be updatable by the system.

Rule 7 : Relational Level Operation

There must be Insert, Delete, Update operations at each level of relations. Set operation like Union, Intersection and minus should also be supported.

Rule 8 : Physical Data Independence

The physical storage of data should not matter to the system. If say, some file supporting table were renamed or moved from one disk to another, it should not effect the application.

Rule 9 : Logical Data Independence

If there is change in the logical structure(table structures) of the database the user view of data should not change. Say, if a table is split into two tables, a new view should give result as the join of the two tables. This rule is most difficult to satisfy.

Rule 10 : Integrity Independence

The database should be able to conforce its own integrity rather than using other programs. Key and Check constraints, trigger etc should be stored in Data Dictionary. This also make **RDBMS** independent of front-end.

Rule 11 : Distribution Independence

A database should work properly regardless of its distribution across a network. This lays foundation of distributed database.

Rule 12 : Nonsubversion rule

If low level access is allowed to a system it should not be able to subvert or bypass integrity rule to change data. This can be achieved by some sort of locking or encryption.

RDBMS Concepts

A **Relational Database management System**(RDBMS) is a database management system based on relational model introduced by E.F Codd. In relational model, data is represented in terms of tuples(rows).

RDBMS is used to manage Relational database. **Relational database** is a collection of organized set of tables from which data can be accessed easily. Relational Database is most commonly used database. It consists of number of tables and each table has its own primary key.

What is Table ?

In Relational database, a **table** is a collection of data elements organized in terms of rows and columns. A table is also considered as convenient representation of **relations**. But a table can have duplicate tuples while a true **relation** cannot have duplicate tuples. Table is the most simplest form of data storage. Below is an example of Employee table.

	Name	Age	Salary
ID			
1	Adam	34	13000
2	Alex	28	15000
3	Stuart	20	18000
4	Ross	42	19020

What is a Record ?

A single entry in a table is called a **Record** or **Row**. A **Record** in a table represents set of related data. For example, the above **Employee** table has 4 records. Following is an example of single record.

1	Adam	34	13000
---	------	----	-------

What is Field ?

A table consists of several records(row), each record can be broken into several smaller entities known as **Fields**. The above **Employee** table consist of four fields, **ID**, **Name**, **Age** and **Salary**.

What is a Column ?

In **Relational** table, a column is a set of value of a particular type. The term **Attribute** is also used to represent a column. For example, in Employee table, Name is a column that represent names of employee.

Name
Adam
Alex
Stuart
Ross

Database Keys

Keys are very important part of Relational database. They are used to establish and identify relation between tables. They also ensure that each record within a table can be uniquely identified by combination of one or more fields within a table.

Super Key

Super Key is defined as a set of attributes within a table that uniquely identifies each record within a table. Super Key is a superset of Candidate key.


Candidate Key

Candidate keys are defined as the set of fields from which primary key can be selected. It is an attribute or set of attribute that can act as a primary key for a table to uniquely identify each record in that table.

Primary Key

Primary key is a candidate key that is most appropriate to become main key of the table. It is a key that uniquely identify each record in a table.

Primary Key



s_id	S_name	age	course	address

Composite Key

Key that consist of two or more attributes that uniquely identify an entity occurrence is called **Composite key**. But any attribute that makes up the **Composite key** is not a simple key in its own.

Composite Key



cust_id	order_id	sale_detail

Secondary or Alternative key

The candidate key which are not selected for primary key are known as secondary keys or alternative keys

Non-key Attribute

Non-key attributes are attributes other than **candidate key** attributes in a table.

Non-prime Attribute

Non-prime Attributes are attributes other than **Primary attribute**.

Normalization of Database

Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like Insertion, Update and Deletion Anamolies. It is a multi-step process that puts data into tabular form by removing duplicated data from the relation tables.

Normalization is used for mainly two purpose,

- Eliminating redundant(useless) data.
- Ensuring data dependencies make sense i.e data is logically stored.

Problem without Normalization

Without Normalization, it becomes difficult to handle and update the database, without facing data loss. Insertion, Updation and Deletion Anamolies are very frequent if Database is not Normalized. To understand these anomalies let us take an example of **Student** table.

S_id	S_Name	S_Address	Subject_opted
401	Adam	Noida	Bio
402	Alex	Panipat	Maths
403	Stuart	Jammu	Maths
404	Adam	Noida	Physics

- **Updation Anamoly** : To update address of a student who occurs twice or more than twice in a table, we will have to update **S_Address** column in all the rows, else data will become inconsistent.
- **Insertion Anamoly** : Suppose for a new admission, we have a Student id(S_id), name and address of a student but if student has not opted for any subjects yet then we have to insert **NULL** there, leading to Insertion Anamoly.
- **Deletion Anamoly** : If (S_id) 401 has only one subject and temporarily he drops it, when we delete that row, entire student record will be deleted along with it.

Normalization Rule

Normalization rule are divided into following normal form.

1. First Normal Form
2. Second Normal Form
3. Third Normal Form
4. BCNF

First Normal Form (1NF)

As per First Normal Form, no two Rows of data must contain repeating group of information i.e each set of column must have a unique value, such that multiple columns cannot be used to fetch the same row. Each table should be organized into rows, and each row should have a primary key that distinguishes it as unique.

The **Primary key** is usually a single column, but sometimes more than one column can be combined to create a single primary key. For example consider a table which is not in First normal form

Student Table :

Student	Age	Subject
Adam	15	Biology, Maths
Alex	14	Maths
Stuart	17	Maths

In First Normal Form, any row must not have a column in which more than one value is saved, like separated with commas. Rather than that, we must separate such data into multiple rows.

Student Table following 1NF will be :

Student	Age	Subject
Adam	15	Biology
Adam	15	Maths
Alex	14	Maths
Stuart	17	Maths

Using the First Normal Form, data redundancy increases, as there will be many columns with same data in multiple rows but each row as a whole will be unique.

Second Normal Form (2NF)

As per the Second Normal Form there must not be any partial dependency of any column on primary key. It means that for a table that has concatenated primary key, each column in the table that is not part of the primary key must depend upon the entire concatenated key for its existence. If any column depends only on one part of the concatenated key, then the table fails **Second normal form**.

In example of First Normal Form there are two rows for Adam, to include multiple subjects that he has opted for. While this is searchable, and follows First normal form, it is an inefficient use of space. Also in the above Table in First Normal Form, while the candidate key is **{Student, Subject}**, **Age** of Student only depends on Student column, which is incorrect as per Second Normal Form. To achieve second normal form, it would be helpful to split out the subjects into an independent table, and match them up using the student names as foreign keys.

New Student Table following 2NF will be :

Student	Age
Adam	15
Alex	14
Stuart	17

In Student Table the candidate key will be **Student** column, because all other column i.e **Age** is dependent on it.

New Subject Table introduced for 2NF will be:

Student	Subject
Adam	Biology
Adam	Maths
Alex	Maths
Stuart	Maths

In Subject Table the candidate key will be {**Student, Subject**} column. Now, both the above tables qualifies for Second Normal Form and will never suffer from Update Anomalies. Although there are a few complex cases in which table in Second Normal Form suffers Update Anomalies, and to handle those scenarios Third Normal Form is there.

Third Normal Form (3NF)

Third Normal form applies that every non-prime attribute of table must be dependent on primary key, or we can say that, there should not be the case that a non-prime attribute is determined by another non-prime attribute. So this transitive functional dependency should be removed from the table and also the table must be in **Second Normal form**. For example, consider a table with following fields.

Student_Detail Table :

Student_id	Student_name	DOB	Street	city	State	Zip
------------	--------------	-----	--------	------	-------	-----

In this table Student_id is Primary key, but street, city and state depends upon Zip. The dependency between zip and other fields is called **transitive dependency**. Hence to apply **3NF**, we need to move the street, city and state to new table, with **Zip** as primary key.

New Student_Detail Table :

Student_id	Student_name	DOB	Zip
------------	--------------	-----	-----

Address Table :

Zip	Street	city	state
-----	--------	------	-------

The advantage of removing transtive dependency is,

- Amount of data duplication is reduced.
- Data integrity achieved.

Boyce and Codd Normal Form (BCNF)

Boyce and Codd Normal Form is a higher version of the Third Normal form. This form deals with certain type of Anamoly that is not handled by 3NF. A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF. For a table to be in BCNF, following conditions must be satisfied:

- R must be in 3rd Normal Form
- and, for each functional dependency ($X \rightarrow Y$), X should be a super Key.

Consider the following relationship : **R (A,B,C,D)**

and following dependencies :

A \rightarrow BCD

BC \rightarrow AD

D \rightarrow B

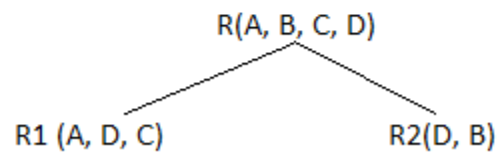
Above relationship is already in 3rd NF. Keys are **A** and **BC**.

Hence, in the functional dependency, **A \rightarrow BCD**, A is the super key.

in second relation, **BC \rightarrow AD**, BC is also a key.

but in, **D \rightarrow B**, D is not a key.

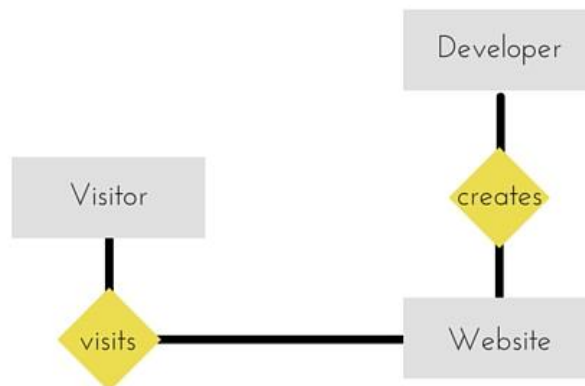
Hence we can break our relationship R into two relationships **R1** and **R2**.



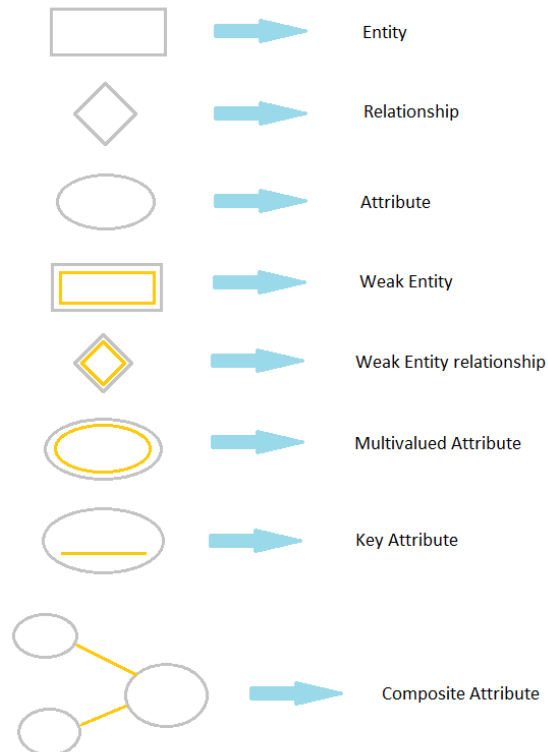
Breaking, table into two tables, one with A, D and C while the other with D and B.

E-R Diagram

ER-Diagram is a visual representation of data that describes how data is related to each other.



Symbols and Notations



Components of E-R Diagram

The E-R diagram has three main components.

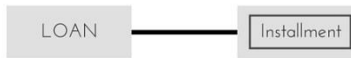
1) Entity

An **Entity** can be any object, place, person or class. In E-R Diagram, an **entity** is represented using rectangles. Consider an example of an Organization. Employee, Manager, Department, Product and many more can be taken as entities from an Organization.



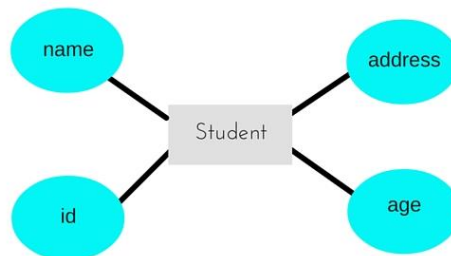
Weak Entity

Weak entity is an entity that depends on another entity. Weak entity doesn't have key attribute of their own. Double rectangle represents weak entity.



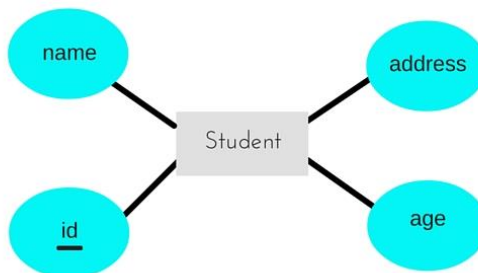
2) Attribute

An **Attribute** describes a property or characteristic of an entity. For example, Name, Age, Address etc can be attributes of a Student. An attribute is represented using ellipse.



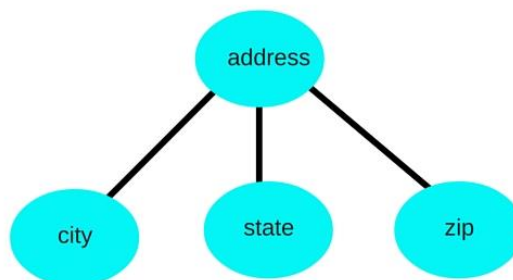
Key Attribute

Key attribute represents the main characteristic of an Entity. It is used to represent Primary key. Ellipse with underlying lines represent Key Attribute.



Composite Attribute

An attribute can also have their own attributes. These attributes are known as **Composite** attribute.



3) Relationship

A Relationship describes relations between **entities**. Relationship is represented using diamonds.



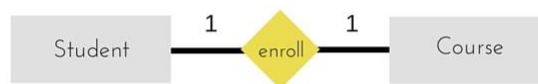
There are three types of relationship that exist between Entities.

- Binary Relationship
- Recursive Relationship
- Ternary Relationship

Binary Relationship

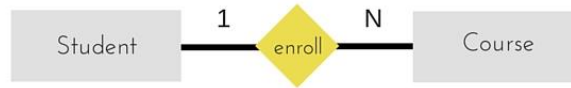
Binary Relationship means relation between two Entities. This is further divided into three types.

1. **One to One** : This type of relationship is rarely seen in real world.



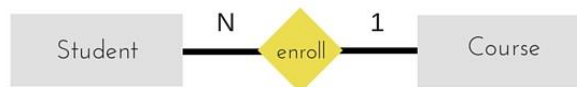
The above example describes that one student can enroll only for one course and a course will also have only one Student. This is not what you will usually see in relationship.

2. **One to Many** : It reflects business rule that one entity is associated with many number of same entity. The example for this relation might sound a little weird, but this means that one student can enroll to many courses, but one course will have one Student.

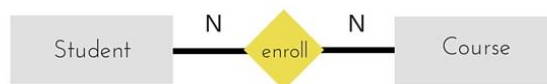


The arrows in the diagram describes that one student can enroll for only one course.

3. **Many to One** : It reflects business rule that many entities can be associated with just one entity. For example, Student enrolls for only one Course but a Course can have many Students.



4. **Many to Many** :



The above diagram represents that many students can enroll for more than one courses.

Recursive Relationship

When an Entity is related with itself it is known as **Recursive** Relationship.

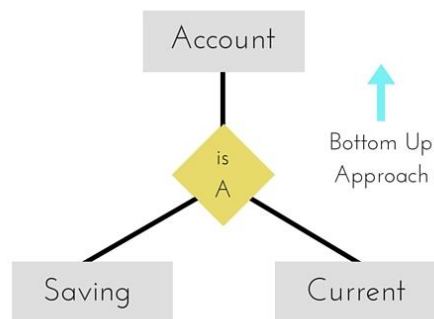


Ternary Relationship

Relationship of degree three is called Ternary relationship.

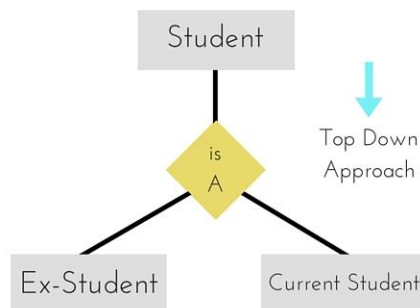
Generalization

Generalization is a bottom-up approach in which two lower level entities combine to form a higher level entity. In generalization, the higher level entity can also combine with other lower level entity to make further higher level entity.



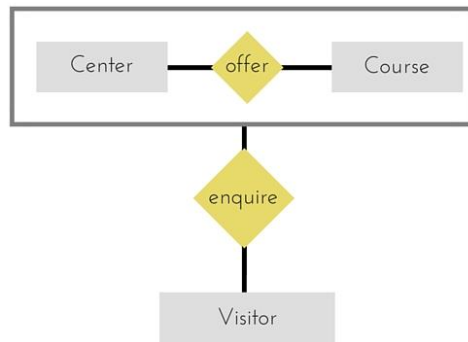
Specialization

Specialization is opposite to Generalization. It is a top-down approach in which one higher level entity can be broken down into two lower level entity. In specialization, some higher level entities may not have lower-level entity sets at all.



Aggregation

Aggregation is a process when relation between two entity is treated as a single entity. Here the relation between Center and Course, is acting as an Entity in relation with Visitor.



SQL tutorial gives unique learning on **Structured Query Language** and it helps to make practice on SQL commands which provides immediate results. SQL is a language of database, it includes database creation, deletion, fetching rows and modifying rows etc.

SQL is an ANSI (American National Standards Institute) standard but there are many different versions of the SQL language.

What is SQL?

SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in relational database.

SQL is the standard language for Relation Database System. All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, postgres and SQL Server use SQL as standard database language.

Also, they are using different dialects, such as:

- MS SQL Server using T-SQL,
- Oracle using PL/SQL,
- MS Access version of SQL is called JET SQL (native format) etc.

Why SQL?

- Allows users to access data in relational database management systems.

- Allows users to describe the data.
- Allows users to define the data in database and manipulate that data.
- Allows to embed within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures, and views

History:

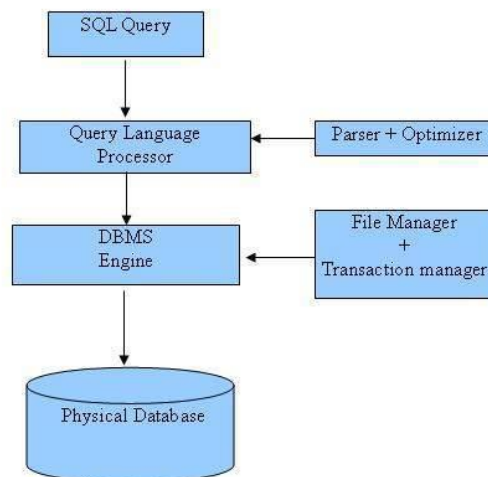
- **1970** -- Dr. Edgar F. "Ted" Codd of IBM is known as the father of relational databases. He described a relational model for databases.
- **1974** -- Structured Query Language appeared.
- **1978** -- IBM worked to develop Codd's ideas and released a product named System/R.
- **1986** -- IBM developed the first prototype of relational database and standardized by ANSI. The first relational database was released by Relational Software and its later becoming Oracle.

SQL Process:

When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task.

There are various components included in the process. These components are Query Dispatcher, Optimization Engines, Classic Query Engine and SQL Query Engine, etc. Classic query engine handles all non-SQL queries but SQL query engine won't handle logical files.

Following is a simple diagram showing SQL Architecture:



The amount of information available to us is literally exploding, and the value of data as an organizational asset is widely recognized. To get the most out of their large and complex datasets, users require tools that simplify the tasks of managing the data and extracting useful information in a timely fashion. Otherwise, data can become a liability, with the cost of

acquiring it and managing it far exceeding the value derived from it. A database management system, or DBMS, is software designed to assist in maintaining and utilizing large collections of data. The collection of data, usually referred to as database, contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient. A **data model** is a collection of high-level data description constructs that hide many low-level storage details. A DBMS allows a user to define the data to be stored in terms of a data model. Most database management systems today are based on the **relational data model**. A relational database consists of a collection of tables (mathematical concept of relation). A row in a table represents a relationship among a set of values. Informally, a table is an entity set, and a row is an entity. The relational model is very simple and elegant: a database is a collection of one or more relations, where each relation is a table with rows and columns. This simple tabular representation enables even novice users to understand the contents of a database, and it permits the use of simple, high-level languages to query the data.

SQL stands for “**Structured Query Language**” and can be pronounced as “SQL” or “sequel – (Structured English Query Language)”. It is a query language used for accessing and modifying information in the database. It has become a Standard Universal Language used by most of the relational database management systems (RDBMS). SQL is tied very closely to the relational model. Few of the SQL commands used in SQL programming are SELECT Statement, UPDATE Statement, INSERT INTO Statement, DELETE Statement, WHERE Clause, ORDER BY Clause, GROUP BY Clause, ORDER Clause, Joins, Views, GROUP Functions, Indexes etc. In a simple manner, SQL is a non-procedural, English-like language that processes data in groups of records rather than one record at a time. Few functions of SQL are:

- store data
- modify data
- retrieve data
- delete data
- create tables and other database objects

Components of SQL

SQL defines following data languages to manipulate data of RDBMS.

DDL : Data Definition Language

All DDL commands are auto-committed. That means it saves all the changes permanently in the database.

Command	Description
create	to create new table or database
alter	for alteration
truncate	delete data from table
drop	to drop a table
rename	to rename a table

DML : Data Manipulation Language

DML commands are not auto-committed. It means changes are not permanent to database, they can be rolled back.

Command	Description
insert	to insert a new row
update	to update existing row
delete	to delete a row
merge	merging two rows or two tables

TCL : Transaction Control Language

These commands are to keep a check on other commands and their affect on the database. These commands can annul changes made by other commands by rolling back to original state. It can also make changes permanent.

Command	Description
commit	to permanently save
rollback	to undo change
savepoint	to save temporarily

DCL : Data Control Language

Data control language provides command to grant and take back authority.

Command	Description
grant	grant permission of right
revoke	take back permission.

DQL : Data Query Language

Command	Description
select	retrieve records from one or more table

Create command

Create is a DDL command used to create a table or a database.

Creating a Database

To create a database in RDBMS, create command is uses. Following is the Syntax,

Create database database-name;

Example for Creating Database

```
create database Test;
```

The above command will create a database named **Test**.

Creating a Table

create command is also used to create a table. We can specify names and datatypes of various columns along. Following is the Syntax,

```
create table table-name  
{  
  column-name1 datatype1,  
  column-name2 datatype2,  
  column-name3 datatype3,  
  column-name4 datatype4  
};
```

create table command will tell the database system to create a new table with given table name and column information.

Example for creating Table

```
create table Student(id int, name varchar, age int);
```

The above command will create a new table **Student** in database system with 3 columns, namely id, name and age.

alter command

alter command is used for alteration of table structures. There are various uses of alter command, such as,

- to add a column to existing table
- to rename any existing column
- to change datatype of any column or to modify its size.
- alter is also used to drop a column.

To Add Column to existing Table

Using alter command we can add a column to an existing table. Following is the Syntax,

```
alter table table-name add(column-name datatype);
```

Here is an Example for this,

```
alter table Student add(address char);
```

The above command will add a new column address to the **Student** table

To Add Multiple Column to existing Table

Using alter command we can even add multiple columns to an existing table. Following is the **Syntax**,

```
alter table table-name add(column-name1 datatype1, column-name2 datatype2, column-name3 datatype3);
```

Here is an Example for this,

```
alter table Student add(father-name varchar(60), mother-name varchar(60), dob date);
```

The above command will add three new columns to the **Student** table

To Add column with Default Value

alter command can add a new column to an existing table with default values. Following is the **Syntax**,

```
alter table table-name add(column-name1 datatype1 default data);
```

Here is an Example for this,

```
alter table Student add(dob date default '1-Jan-99');
```

The above command will add a new column with default value to the **Student** table

To Modify an existing Column

alter command is used to modify data type of an existing column . Following is the **Syntax**,

```
alter table table-name modify(column-name datatype);
```

Here is an Example for this,

```
alter table Student modify(address varchar(30));
```

The above command will modify address column of the **Student table**

To Rename a column

Using alter command you can rename an existing column. Following is the Syntax,

```
alter table table-name rename old-column-name to column-name;
```

Here is an Example for this,

```
alter table Student rename address to Location;
```

The above command will rename address column to Location.

To Drop a Column

alter command is also used to drop columns also. Following is the Syntax,

```
alter table table-name drop(column-name);
```

Here is an Example for this,

```
alter table Student drop(address);
```

The above command will drop address column from the **Student table**

SQL queries to Truncate, Drop or Rename a Table

truncate command

truncate command removes all records from a table. But this command will not destroy the table's structure. When we apply truncate command on a table its Primary key is initialized. Following is its Syntax,

```
truncate table table-name
```

Here is an Example explaining it.

```
truncate table Student;
```

The above query will delete all the records of **Student** table.

truncate command is different from **delete** command. delete command will delete all the rows from a table whereas truncate command re-initializes a table(like a newly created table).

For eg. If you have a table with 10 rows and an auto_increment primary key, if you use delete command to delete all the rows, it will delete all the rows, but will not initialize the

primary key, hence if you will insert any row after using delete command, the auto_increment primary key will start from 11. But in case of truncate command, primary key is re-initialized.

drop command

drop query completely removes a table from database. This command will also destroy the table structure. Following is its Syntax,

```
drop table table-name
```

Here is an Example explaining it.

```
drop table Student;
```

The above query will delete the **Student** table completely. It can also be used on Databases. For Example, to drop a database,

```
drop database Test;
```

The above query will drop a database named **Test** from the system.

rename query

rename command is used to rename a table. Following is its Syntax,

```
rename table old-table-name to new-table-name
```

Here is an Example explaining it.

```
rename table Student to Student-record;
```

The above query will rename **Student** table to **Student-record**.

DML command

Data Manipulation Language (DML) statements are used for managing data in database. DML commands are not auto-committed. It means changes made by DML command are not permanent to database, it can be rolled back.

1) INSERT command

Insert command is used to insert data into a table. Following is its general syntax,

```
INSERT into table-name values(data1,data2,..)
```

Lets see an example,

Consider a table **Student** with following fields.

S_id	S_Name	age
------	--------	-----

```
INSERT into Student values(101,'Adam',15);
```

The above command will insert a record into **Student** table.

S_id	S_Name	age
101	Adam	15

Example to Insert NULL value to a column

Both the statements below will insert NULL value into **age** column of the Student table.

```
INSERT into Student(id,name) values(102,'Alex');
```

Or,

```
INSERT into Student values(102,'Alex',null);
```

The above command will insert only two column value other column is set to null.

S_id	S_Name	age
101	Adam	15
102	Alex	

Example to Insert Default value to a column

```
INSERT into Student values(103,'Chris',default)
```

S_id	S_Name	age
101	Adam	15
102	Alex	
103	chris	14

Suppose the **age** column of student table has default value of 14.

Also, if you run the below query, it will insert default value into the age column, whatever the default value may be.

```
INSERT into Student values(103,'Chris')
```

2) UPDATE command

Update command is used to update a row of a table. Following is its general syntax,

```
UPDATE table-name set column-name = value where condition;
```

Lets see an example,

```
update Student set age=18 where s_id=102;
```

S_id	S_Name	age
101	Adam	15
102	Alex	18
103	chris	14

Example to Update multiple columns

```
UPDATE Student set s_name='Abhi',age=17 where s_id=103;
```

The above command will update two columns of a record.

S_id	S_Name	age
101	Adam	15
102	Alex	18
103	Abhi	17

3) Delete command

Delete command is used to delete data from a table. Delete command can also be used with condition to delete a particular row. Following is its general syntax,

```
DELETE from table-name;
```

Example to Delete all Records from a Table

```
DELETE from Student;
```

The above command will delete all the records from **Student** table.

Example to Delete a particular Record from a Table

Consider the following **Student** table

S_id	S_Name	age
101	Adam	15
102	Alex	18
103	Abhi	17

```
DELETE from Student where s_id=103;
```

The above command will delete the record where s_id is 103 from **Student** table.

S_id	S_Name	age
101	Adam	15
102	Alex	18

TCL command

Transaction Control Language(TCL) commands are used to manage transactions in database. These are used to manage the changes made by DML statements. It also allows statements to be grouped together into logical transactions.

Commit command

Commit command is used to permanently save any transaction into database.

Following is Commit command's syntax,

```
commit;
```

Rollback command

This command restores the database to last committed state. It is also use with savepoint command to jump to a savepoint in a transaction.

Following is Rollback command's syntax,

```
rollback to savepoint-name;
```

Savepoint command

savepoint command is used to temporarily save a transaction so that you can rollback to that point whenever necessary.

Following is savepoint command's syntax,

```
savepoint savepoint-name;
```

Example of Savepoint and Rollback

Following is the **class** table,

ID	NAME
1	abhi
2	adam
4	alex

Lets use some SQL queries on the above table and see the results.

```
INSERT into class values(5,'Rahul');
commit;
UPDATE class set name='abhijit' where id='5';
savepoint A;
INSERT into class values(6,'Chris');
savepoint B;
INSERT into class values(7,'Bravo');
savepoint C;
SELECT * from class;
```

The resultant table will look like,

ID	NAME
1	abhi
2	adam
4	alex
5	abhijit
6	chris
7	bravo

Now **rollback** to **savepoint B**

```
rollback to B;
SELECT * from class;
```

The resultant table will look like

ID	NAME
1	abhi
2	adam
4	alex
5	abhijit
6	chris

Now **rollback** to **savepoint A**

```
rollback to A;
SELECT * from class;
```

The result table will look like

ID	NAME
----	------

1	abhi
2	adam
4	alex
5	abhijit

DCL command

Data Control Language(DCL) is used to control privilege in Database. To perform any operation in the database, such as for creating tables, sequences or views we need privileges. Privileges are of two types,

- **System** : creating session, table etc are all types of system privilege.
- **Object** : any command or query to work on tables comes under object privilege.

DCL defines two commands,

- **Grant** : Gives user access privileges to database.
- **Revoke** : Take back permissions from user.

To Allow a User to create Session

```
grant create session to username;
```

To Allow a User to create Table

```
grant create table to username;
```

To provide User with some Space on Tablespace to store Table

```
alter user username quota unlimited on system;
```

To Grant all privilege to a User

```
grant sysdba to username
```

To Grant permission to Create any Table

grant create any table to username

To Grant permission to Drop any Table

grant drop any table to username

To take back Permissions

revoke create table from username

WHERE clause

Where clause is used to specify condition while retrieving data from table. Where clause is used mostly with Select, Update and Delete query. If condition specified by where clause is true then only the result from table is returned.

Syntax for WHERE clause

```
SELECT column-name1,  
column-name2,  
column-name3,  
column-nameN  
from table-name WHERE [condition];
```

Example using WHERE clause

Consider a **Student** table,

s_id	s_Name	age	address
101	Adam	15	Noida
102	Alex	18	Delhi
103	Abhi	17	Rohtak
104	Ankit	22	Panipat

Now we will use a SELECT statement to display data of the table, based on a condition, which we will add to the SELECT query using WHERE clause.

```
SELECT s_id,  
s_name,  
age,  
address  
from Student WHERE s_id=101;
```

s_id	s_Name	age	address
101	Adam	15	Noida

SELECT Query

Select query is used to retrieve data from a tables. It is the most used SQL query. We can retrieve complete tables, or partial by mentioning conditions using WHERE clause.

Syntax of SELECT Query

```
SELECT column-name1, column-name2, column-name3, column-nameN from table-name;
```

Example for SELECT Query

Consider the following **Student** table,

S_id	S_Name	age	address
101	Adam	15	Noida
102	Alex	18	Delhi
103	Abhi	17	Rohtak
104	Ankit	22	Panipat

```
SELECT s_id, s_name, age from Student.
```

The above query will fetch information of s_id, s_name and age column from Student table

S_id	S_Name	age
101	Adam	15
102	Alex	18
103	Abhi	17
104	Ankit	22

Example to Select all Records from Table

A special character **asterisk** * is used to address all the data(belonging to all columns) in a query. SELECT statement uses * character to retrieve all records from a table.

```
SELECT * from student;
```

The above query will show all the records of Student table, that means it will show complete Student table as result.

S_id	S_Name	age	address
101	Adam	15	Noida
102	Alex	18	Delhi
103	Abhi	17	Rohtak
104	Ankit	22	Panipat

Example to Select particular Record based on Condition

```
SELECT * from Student WHERE s_name = 'Abhi';
```

103	Abhi	17	Rohtak
-----	------	----	--------

Example to Perform Simple Calculations using Select Query

Consider the following **Employee** table.

eid	Name	age	salary
101	Adam	26	5000
102	Ricky	42	8000
103	Abhi	22	10000
104	Rohan	35	5000

```
SELECT eid, name, salary+3000 from Employee;
```

The above command will display a new column in the result, showing 3000 added into existing salaries of the employees.

eid	Name	salary+3000
101	Adam	8000
102	Ricky	11000
103	Abhi	13000
104	Rohan	8000

Like clause

Like clause is used as condition in SQL query. **Like** clause compares data with an expression using wildcard operators. It is used to find similar data from the table.

Wildcard operators

There are two wildcard operators that are used in like clause.

- **Percent sign %**: represents zero, one or more than one character.
- **Underscore sign _**: represents only one character.

Example of LIKE clause

Consider the following **Student** table.

s_id	s_Name	age
101	Adam	15
102	Alex	18
103	Abhi	17

SELECT * from Student where s_name like 'A%';

The above query will return all records where **s_name** starts with character 'A'.

s_id	s_Name	age
101	Adam	15
102	Alex	18
103	Abhi	17

Example

SELECT * from Student where s_name like '_d%';

The above query will return all records from **Student** table where **s_name** contain 'd' as second character.

s_id	s_Name	age
101	Adam	15

Example

SELECT * from Student where s_name like '%x';

The above query will return all records from **Student** table where **s_name** contain 'x' as last character.

s_id	s_Name	age
102	Alex	18

Order By Clause

Order by clause is used with **Select** statement for arranging retrieved data in sorted order. The **Order by** clause by default sort data in ascending order. To sort data in descending order **DESC** keyword is used with **Order by** clause.

Syntax of Order By

```
SELECT column-list|* from table-name order by asc|desc;
```

Example using Order by

Consider the following **Emp** table,

eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

```
SELECT * from Emp order by salary;
```

The above query will return result in ascending order of the **salary**.

eid	name	age	salary
403	Rohan	34	6000
402	Shane	29	8000
405	Tiger	35	8000
401	Anu	22	9000
404	Scott	44	10000

Example of Order by DESC

Consider the **Emp** table described above,

```
SELECT * from Emp order by salary DESC;
```

The above query will return result in descending order of the **salary**.

eid	name	age	salary
404	Scott	44	10000
401	Anu	22	9000
405	Tiger	35	8000
402	Shane	29	8000
403	Rohan	34	6000

Group By Clause

Group by clause is used to group the results of a SELECT query based on one or more columns. It is also used with SQL functions to group the result from one or more tables.

Syntax for using Group by in a statement.

```
SELECT column_name, function(column_name)
FROM table_name
WHERE condition
GROUP BY column_name
```

Example of Group by in a Statement

Consider the following **Emp** table.

eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	9000
405	Tiger	35	8000

Here we want to find name and age of employees grouped by their salaries

SQL query for the above requirement will be,

```
SELECT name, age
from Emp group by salary
```

Result will be,

name	age
Rohan	34
shane	29
anu	22

Example of Group by in a Statement with WHERE clause

Consider the following **Emp** table

eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000

403	Rohan	34	6000
404	Scott	44	9000
405	Tiger	35	8000

SQL query will be,

```
select name, salary
from Emp
where age > 25
group by salary
```

Result will be.

name	salary
Rohan	6000
Shane	8000
Scott	9000

You must remember that Group By clause will always come at the end, just like the Order by clause.

HAVING Clause

having clause is used with SQL Queries to give more precise condition for a statement. It is used to mention condition in Group based SQL functions, just like WHERE clause.

Syntax for having will be,

```
select column_name, function(column_name)
FROM table_name
WHERE column_name condition
GROUP BY column_name
HAVING function(column_name) condition
```

Example of HAVING Statement

Consider the following **Sale** table.

oid	order_name	previous_balance	customer
11	ord1	2000	Alex

12	ord2	1000	Adam
13	ord3	2000	Abhi
14	ord4	1000	Adam
15	ord5	2000	Alex

Suppose we want to find the customer whose previous_balance sum is more than 3000.

We will use the below SQL query,

```
SELECT *
from sale group customer
having sum(previous_balance) > 3000
```

Result will be,

oid	order_name	previous_balance	customer
11	ord1	2000	Alex

Distinct keyword

The **distinct** keyword is used with **Select** statement to retrieve unique values from the table. **Distinct** removes all the duplicate records while retrieving from database.

Syntax for DISTINCT Keyword

```
SELECT distinct column-name from table-name;
```

Example

Consider the following **Emp** table.

eid	name	age	salary
401	Anu	22	5000
402	Shane	29	8000
403	Rohan	34	10000
404	Scott	44	10000
405	Tiger	35	8000

```
select distinct salary from Emp;
```

The above query will return only the unique salary from **Emp** table

salary
5000
8000
10000

AND & OR operator

AND and **OR** operators are used with **Where** clause to make more precise conditions for fetching data from database by combining more than one condition together.

AND operator

AND operator is used to set multiple conditions with Where clause.

Example of AND

Consider the following **Emp** table

eid	name	age	salary
401	Anu	22	5000
402	Shane	29	8000
403	Rohan	34	12000
404	Scott	44	10000
405	Tiger	35	9000

```
SELECT * from Emp WHERE salary < 10000 AND age > 25
```

The above query will return records where salary is less than 10000 and age greater than 25.

eid	name	age	salary
402	Shane	29	8000
405	Tiger	35	9000

OR operator

OR operator is also used to combine multiple conditions with Where clause. The only difference between AND and OR is their behaviour. When we use AND to combine two or more than two conditions, records satisfying all the condition will be in the result. But in case of OR, atleast one condition from the conditions specified must be satisfied by any record to be in the result.

Example of OR

Consider the following **Emp** table

eid	name	age	salary
401	Anu	22	5000
402	Shane	29	8000
403	Rohan	34	12000
404	Scott	44	10000
405	Tiger	35	9000

```
SELECT * from Emp WHERE salary > 10000 OR age > 25
```

The above query will return records where either salary is greater than 10000 or age greater than 25.

402	Shane	29	8000
403	Rohan	34	12000
404	Scott	44	10000
405	Tiger	35	9000

The BETWEEN operator is used to select values within a range.

The SQL BETWEEN Operator

The BETWEEN operator selects values within a range. The values can be numbers, text, or dates.

SQL BETWEEN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

Demo Database

Below is a selection from the "Products" table:

ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	1	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	1	2	36 boxes	21.35

BETWEEN Operator Example

The following SQL statement selects all products with a price BETWEEN 10 and 20:

Example

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;
```

NOT BETWEEN Operator Example

To display the products outside the range of the previous example, use NOT BETWEEN:

Example

```
SELECT * FROM Products
WHERE Price NOT BETWEEN 10 AND 20;
```

The IN Operator

The IN operator allows you to specify multiple values in a WHERE clause.

SQL IN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1,value2,...);
```

IN Operator Example

The following SQL statement selects all customers with a City of "Paris" or "London":

Example

```
SELECT * FROM Customers
WHERE City IN ('Paris','London');
```

Aggregate Functions:

SQL has many built-in functions for performing calculations on data.

SQL Aggregate Functions

SQL aggregate functions return a single value, calculated from values in a column.

Useful aggregate functions:

- AVG() - Returns the average value
- COUNT() - Returns the number of rows

- FIRST() - Returns the first value
- LAST() - Returns the last value
- MAX() - Returns the largest value
- MIN() - Returns the smallest value
- SUM() - Returns the sum

SQL Scalar functions

SQL scalar functions return a single value, based on the input value.

Useful scalar functions:

- UCASE() - Converts a field to upper case
- LCASE() - Converts a field to lower case
- MID() - Extract characters from a text field
- LEN() - Returns the length of a text field
- ROUND() - Rounds a numeric field to the number of decimals specified
- NOW() - Returns the current system date and time
- FORMAT() - Formats how a field is to be displayed

The AVG() Function

The AVG() function returns the average value of a numeric column.

SQL AVG() Syntax

```
SELECT AVG(column_name) FROM table_name
```

SQL AVG() Example

The following SQL statement gets the average value of the "Price" column from the "Products" table:

Example

```
SELECT AVG(Price) AS PriceAverage FROM Products;
```

The COUNT() function returns the number of rows that matches a specified criteria.

SQL COUNT(column_name) Syntax

The COUNT(column_name) function returns the number of values (NULL values will not be counted) of the specified column:

```
SELECT COUNT(column_name) FROM table_name;
```

SQL COUNT(*) Syntax

The COUNT(*) function returns the number of records in a table:

```
SELECT COUNT(*) FROM table_name;
```

SQL COUNT(DISTINCT column_name) Syntax

The COUNT(DISTINCT column_name) function returns the number of distinct values of the specified column:

```
SELECT COUNT(DISTINCT column_name) FROM table_name;
```

Note: COUNT(DISTINCT) works with ORACLE and Microsoft SQL Server, but not with Microsoft Access.

Demo Database

Below is a selection from the "Orders" table:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10265	7	2	1996-07-25	1
10266	87	3	1996-07-26	3
10267	25	4	1996-07-29	1

SQL COUNT(column_name) Example

The following SQL statement counts the number of orders from "CustomerID"=7 from the "Orders" table:

Example

```
SELECT COUNT(CustomerID) AS OrdersFromCustomerID7 FROM Orders
WHERE CustomerID=7;
```

SQL COUNT(*) Example

The following SQL statement counts the total number of orders in the "Orders" table:

Example

```
SELECT COUNT(*) AS NumberOfOrders FROM Orders;
```

SQL COUNT(DISTINCT column_name) Example

The following SQL statement counts the number of unique customers in the "Orders" table:

Example

```
SELECT COUNT(DISTINCT CustomerID) AS NumberOfCustomers FROM Orders;
```

The FIRST() Function

The FIRST() function returns the first value of the selected column.

SQL FIRST() Syntax

```
SELECT FIRST(column_name) FROM table_name;
```

SQL FIRST() Example

The following SQL statement selects the first value of the "CustomerName" column from the "Customers" table:

Example

```
SELECT FIRST(CustomerName) AS FirstCustomer FROM Customers;
```

The LAST() Function

The LAST() function returns the last value of the selected column.

SQL LAST() Syntax

```
SELECT LAST(column_name) FROM table_name;
```

SQL LAST() Example

The following SQL statement selects the last value of the "CustomerName" column from the "Customers" table:

Example

```
SELECT LAST(CustomerName) AS LastCustomer FROM Customers;
```

The MAX() Function

The MAX() function returns the largest value of the selected column.

SQL MAX() Syntax

```
SELECT MAX(column_name) FROM table_name;
```

SQL MAX() Example

The following SQL statement gets the largest value of the "Price" column from the "Products" table:

Example

```
SELECT MAX(Price) AS HighestPrice FROM Products;
```

The MIN() Function

The MIN() function returns the smallest value of the selected column.

SQL MIN() Syntax

```
SELECT MIN(column_name) FROM table_name;
```

SQL MIN() Example

The following SQL statement gets the smallest value of the "Price" column from the "Products" table:

Example

```
SELECT MIN(Price) AS SmallestOrderPrice FROM Products;
```

The SUM() Function

The SUM() function returns the total sum of a numeric column.

SQL SUM() Syntax

```
SELECT SUM(column_name) FROM table_name;
```

Demo Database

Below is a selection from the "OrderDetails" table:

OrderDetailID	OrderID	ProductID	Quantity
1	10248	11	12
2	10248	42	10
3	10248	72	5
4	10249	14	9
5			

SQL SUM() Example

The following SQL statement finds the sum of all the "Quantity" fields for the "OrderDetails" table:

Example

```
SELECT SUM(Quantity) AS TotalItemsOrdered FROM OrderDetails;
```

The UCASE() Function

The UCASE() function converts the value of a field to uppercase.

SQL UCASE() Syntax

```
SELECT UCASE(column_name) FROM table_name;
```

SQL UCASE() Example

The following SQL statement selects the "CustomerName" and "City" columns from the "Customers" table, and converts the "CustomerName" column to uppercase:

Example

```
SELECT UCASE(CustomerName) AS Customer, City  
FROM Customers;
```

The LCASE() Function

The LCASE() function converts the value of a field to lowercase.

SQL LCASE() Syntax

```
SELECT LCASE(column_name) FROM table_name;
```

SQL LCASE() Example

The following SQL statement selects the "CustomerName" and "City" columns from the "Customers" table, and converts the "CustomerName" column to lowercase:

Example

```
SELECT LCASE(CustomerName) AS Customer, City  
FROM Customers;
```

The LEN() Function

The LEN() function returns the length of the value in a text field.

SQL LEN() Syntax

```
SELECT LEN(column_name) FROM table_name;
```


SQL LEN() Example

The following SQL statement selects the "CustomerName" and the length of the values in the "Address" column from the "Customers" table:

Example

```
SELECT CustomerName,LEN(Address) as LengthOfAddress  
FROM Customers;
```

The ROUND() Function

The ROUND() function is used to round a numeric field to the number of decimals specified.

SQL ROUND() Syntax

```
SELECT ROUND(column_name,decimals) FROM table_name;
```

SQL ROUND() Example

The following SQL statement selects the product name and rounds the price in the "Products" table:

Example

```
SELECT ProductName, ROUND(Price,0) AS RoundedPrice  
FROM Products;
```

The NOW() Function

The NOW() function returns the current system date and time.

SQL NOW() Syntax

```
SELECT NOW() FROM table_name;
```

SQL NOW() Example

The following SQL statement selects the product name, and price for today from the "Products" table:

Example

```
SELECT ProductName, Price, Now() AS PerDate  
FROM Products;
```

SQL Constraints

SQL Constraints are rules used to limit the type of data that can go into a table, to maintain the accuracy and integrity of the data inside table.

Constraints can be divided into following two types,

- **Column level constraints** : limits only column data
- **Table level constraints** : limits whole table data

Constraints are used to make sure that the integrity of data is maintained in the database. Following are the most used constraints that can be applied to a table.

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

NOT NULL Constraint

NOT NULL constraint restricts a column from having a NULL value. Once **NOT NULL** constraint is applied to a column, you cannot pass a null value to that column. It enforces a column to contain a proper value. One important point to note about NOT NULL constraint is that it cannot be defined at table level.

Example using NOT NULL constraint

```
CREATE table Student(s_id int NOT NULL, Name varchar(60), Age int);
```

The above query will declare that the **s_id** field of **Student** table will not take NULL value.

UNIQUE Constraint

UNIQUE constraint ensures that a field or column will only have unique values. A UNIQUE constraint field will not have duplicate data. UNIQUE constraint can be applied at column level or table level.

Example using UNIQUE constraint when creating a Table (Table Level)

```
CREATE table Student(s_id int NOT NULL UNIQUE, Name varchar(60), Age int);
```

The above query will declare that the **s_id** field of **Student** table will only have unique values and wont take NULL value.

Example using UNIQUE constraint after Table is created (Column Level)

```
ALTER table Student add UNIQUE(s_id);
```

The above query specifies that **s_id** field of **Student** table will only have unique value.

Primary Key Constraint

Primary key constraint uniquely identifies each record in a database. A Primary Key must contain unique value and it must not contain null value. Usually Primary Key is used to index the data inside the table.

Example using PRIMARY KEY constraint at Table Level

```
CREATE table Student (s_id int PRIMARY KEY, Name varchar(60) NOT NULL, Age int);
```

The above command will creates a PRIMARY KEY on the **s_id**.

Example using PRIMARY KEY constraint at Column Level

```
ALTER table Student add PRIMARY KEY (s_id);
```

The above command will creates a PRIMARY KEY on the **s_id**.

Foreign Key Constraint

FOREIGN KEY is used to relate two tables. FOREIGN KEY constraint is also used to restrict actions that would destroy links between tables. To understand FOREIGN KEY, let's see it using two table.

Customer_Detail Table :

c_id	Customer_Name	address
101	Adam	Noida
102	Alex	Delhi
103	Stuart	Rohtak

Order_Detail Table :

Order_id	Order_Name	c_id
10	Order1	101
11	Order2	103
12	Order3	102

In **Customer_Detail** table, c_id is the primary key which is set as foreign key in **Order_Detail** table. The value that is entered in c_id which is set as foreign key in **Order_Detail** table must be present in **Customer_Detail** table where it is set as primary key. This prevents invalid data to be inserted into c_id column of **Order_Detail** table.

Example using FOREIGN KEY constraint at Table Level

```
CREATE table Order_Detail(order_id int PRIMARY KEY,  
order_name varchar(60) NOT NULL,  
c_id int FOREIGN KEY REFERENCES Customer_Detail(c_id));
```

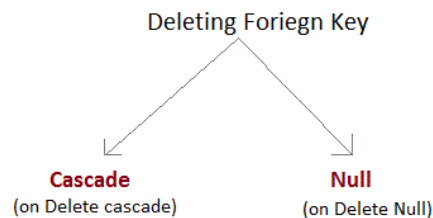
In this query, c_id in table Order_Detail is made as foreign key, which is a reference of c_id column of Customer_Detail.

Example using FOREIGN KEY constraint at Column Level

```
ALTER table Order_Detail add FOREIGN KEY (c_id) REFERENCES Customer_Detail(c_id);
```

Behaviour of Foreign Key Column on Delete

There are two ways to maintain the integrity of data in Child table, when a particular record is deleted in main table. When two tables are connected with Foreign key, and certain data in the main table is deleted, for which record exist in child table too, then we must have some mechanism to save the integrity of data in child table.



- **On Delete Cascade** : This will remove the record from child table, if that value of foreign key is deleted from the main table.

- **On Delete Null** : This will set all the values in that record of child table as NULL, for which the value of foreign key is deleted from the main table.
- If we don't use any of the above, then we cannot delete data from the main table for which data in child table exists. We will get an error if we try to do so.

ERROR : Record in child table exist

CHECK Constraint

CHECK constraint is used to restrict the value of a column between a range. It performs check on the values, before storing them into the database. Its like condition checking before saving data into a column.

Example using CHECK constraint at Table Level

```
create table Student(s_id int NOT NULL CHECK(s_id > 0),  
Name varchar(60) NOT NULL,  
Age int);
```

The above query will restrict the s_id value to be greater than zero.

Example using CHECK constraint at Column Level

```
ALTER table Student add CHECK(s_id > 0);
```