UNIT I	RELATIONAL DATABASES
Purpose of Database System – Views of data – Data Models – Database System Architecture – Introduction to relational databases – Relational Model – Keys – Relational Algebra – SQL fundamentals – Advanced SQL features – Embedded SQL– Dynamic SQL	

INTRODUCTION

DATABASE

Database is collection of data which is related by some aspect. Data is collection of facts and figures which can be processed to produce information. Mostly data represents recordable facts. Data aids in producing information which is based on facts. A database management system stores data, in such a way which is easier to retrieve, manipulate and helps to produce information.

So a database is a collection of related data that we can use for

- Defining - specifying types of data
- Constructing - storing & populating
- Manipulating - querying, updating, reporting

DISADVANTAGES OF FILE SYSTEM OVER DB

In the early days, File-Processing system is used to store records. It uses various files for storing the records.

Drawbacks of using file systems to store data:

- Data redundancy and inconsistency
  -Multiple file formats, duplication of information in different files
- Difficulty in accessing data
  Need to write a new program to carry out each new task
- Data isolation — multiple files and formats
- Integrity problems
  - Hard to add new constraints or change existing ones
- Atomicity problem
  -Failures may leave database in an inconsistent state with partial updates carried Out.
  E.g. transfer of funds from one account to another should either complete or not happen at all
- Concurrent access anomalies
  - Concurrent accessed needed for performance
- Security problems

Database systems offer solutions to all the above problems

PURPOSE OF DATABASE SYSTEM

The typical file processing system is supported by a conventional operating system. The system stores permanent records in various files, and it needs different application programs to extract records from, and add records to, the appropriate files. A file processing system has a number of major disadvantages.

- Data redundancy and inconsistency
- Difficulty in accessing data
- Data isolation – multiple files and formats
- Integrity problems
- Atomicity of updates
- Concurrent access by multiple users
- Security problems

1.Data redundancy and inconsistency:

In file processing, every user group maintains its own files for handling its data processing applications.

Example:

Consider the UNIVERSITY database. Here, two groups of users might be the course registration personnel and the accounting office. The accounting office also keeps data on registration and related billing information, whereas the registration office keeps track of student courses and grades.Storing the same data multiple times is called data redundancy.This redundancy leads to several problems.

•Need to perform a single logical update multiple times.

•Storage space is wasted.

•Files that represent the same data may become inconsistent.

Data inconsistency is the various copies of the same data may no larger Agree. **Example:**

One user group may enter a student's birth date erroneously as JAN-19-1984, whereas the other user groups may enter the correct value of JAN-29-1984.

2. Difficulty in accessing data
File processing environments do not allow needed data to be retrieved in a convenient and efficient manner.

3.Data isolation
Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.

4.Integrity problems
The data values stored in the database must satisfy certain types of consistency constraints. **Example:**
The balance of certain types of bank accounts may never fall below a prescribed amount . Developers enforce these constraints in the system by addition appropriate code in the various application programs

5.Atomicity problems
Atomic means the transaction must happen in its entirety or not at all. It is difficult to ensure atomicity in a conventional file processing system.

Example:
Consider a program to transfer $50 from account A to account B. If a system failure occurs during the execution of the program, it is possible that the $50 was removed from account A but was not credited to account B, resulting in an inconsistent database state.

6.Concurrent access anomalies
For the sake of overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously. In such an environment, interaction of concurrent updates is possible and may result in inconsistent data. To guard against this possibility, the system must maintain some form of supervision. But supervision is difficult to
provide because data may be accessed by many different application programs that have not been coordinated previously.

Example: When several reservation clerks try to assign a seat on an airline flight, the system should ensure that each seat can be accessed by only one clerk at a time for assignment to a passenger.

7. Security problems
Enforcing security constraints to the file processing system is difficult.

APPLICATION OF DATABASE

Database Applications

- Banking: all transactions
- Airlines: reservations, schedules
- Universities: registration, grades
- Sales: customers, products, purchases
- Manufacturing: production, inventory, orders, supply chain
- Human resources: employee records, salaries, tax deductions
- Telecommunication: Call History, Billing
- Credit card transactions: Purchase details,Statements

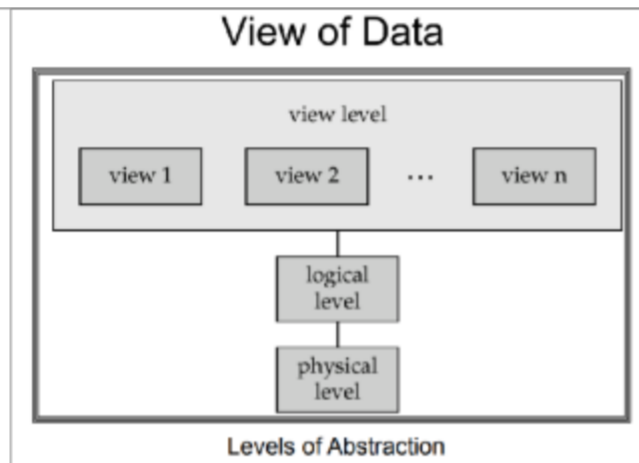
VIEWS OF DATA
It refers that how database is actually stored in database, what data and structure of data used by database for data.
So describe all this database provides user with views and these are

- **Data abstraction**
- **Instances and schemas**

Data abstraction
As a data in database are stored with very complex data structure so when user come and want to access any data, he will not be able to access data if he has go through this data structure. So to simplify the interaction of user and database, DBMS hides some information which is not of user interest, a this is called data abstraction:- **So developer hides complexity from user and store abstract view of data.**
Data abstraction has three level of abstractions

- **level / internal level**
- **Logical level / conceptual level**
- **view level / external level**

Physical level:- this is the lowest level of data abstraction which describe How data is actual stored in database. This level basically describe the data structure and access path /indexing use for accessing file.
Logical level:- The next level of abstraction describe what data are stored in the database and what are the relationship existed among those of data.
View level:- In this level user only interact with database and the complexity remain unview . user see data and there may be many views of one data like chart and graph.



DATA MODELS IN DBMS

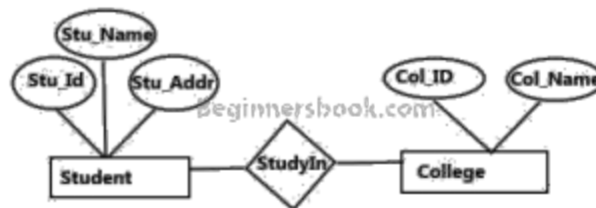
A **Data Model** is a logical structure of Database. It describes the design of database to reflect entities, attributes, relationship among data, constrains etc.

Types of Data Models:

Object based logical Models – Describe data at the conceptual and view levels.

1. E-R Model

An **entity–relationship model (ER model)** is a systematic way of describing and defining a business process. An ER model is typically implemented as a database. The main components of E-R model are: entity set and relationship set.



Sample E-R Diagram

2. **Object oriented Model**

An object data model is a data model based on object-oriented programming, associating methods (procedures) with objects that can benefit from class hierarchies. Thus, ―objects‖ are levels of abstraction that include attributes and behavior

Record based logical Models – Like Object based model, they also describe data at the conceptual and view levels. These models specify logical structure of database with records, fields and attributes.

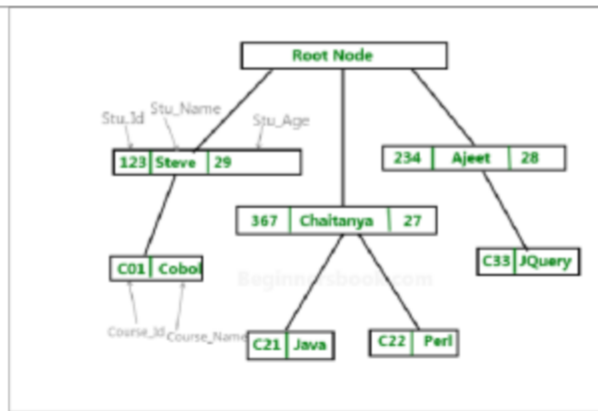
1. **Relational Model**

In relational model, the data and relationships are represented by collection of inter-related tables. Each table is a group of column and rows, where column represents attribute of an entity and rows represents records. Sample relationship Model: Student table with 3 columns and three records.

Stu_Id	Stu_Name	Stu_Age
111	Ashish	23
123	Saurav	22
169	Lester	24

2. **Hierarchical Model**

In hierarchical model, data is organized into a tree like structure with each record is having one parent record and many children. The main drawback of this model is that, it can have only one to many relationships between nodes. Sample Hierarchical Model Diagram:



3. **Network Model** – Network Model is same as hierarchical model except that it has graph-like structure rather than a tree-based structure. Unlike hierarchical model, this model allows each record to have more than one parent record.

Physical Data Models – These models describe data at the lowest level of abstraction.

Three Schema Architecture

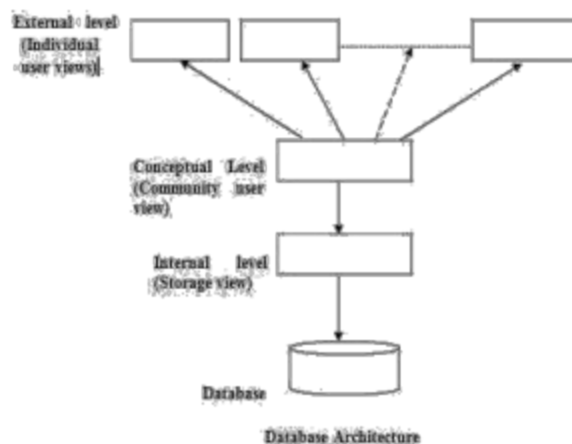
The goal of the three schema architecture is to separate the user applications and the physical database. The schemas can be defined at the following levels:

1. **The internal level** – has an internal schema which describes the physical storage structure of the database. Uses a physical data model and describes the complete details of data storage and access paths for the database.

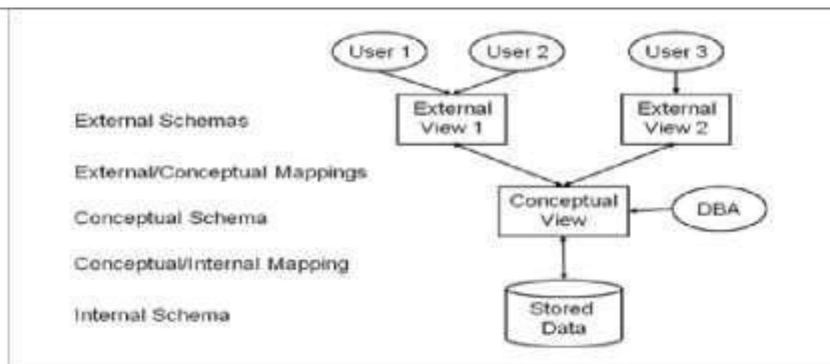
2. **The conceptual level** – has a conceptual schema which describes the structure of the database for users. It hides the details of the physical storage structures, and concentrates on describing entities, data types, relationships, user operations and constraints. Usually a representational data model is used to describe the conceptual schema.

3. **The External or View level** – includes external schemas or user vies. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. Represented using the representational data model.

The three schema architecture is used to visualize the schema levels in a database. The three schemas are only descriptions of data, the data only actually exists is at the physical level.



Database Architecture



COMPONENTS OF DBMS

Database Users

Users are differentiated by the way they expect to interact with the system

- Application programmers
- Sophisticated users
- Naïve users
- Database Administrator
- Specialized users etc,.

Application programmers:

Professionals who write application programs and using these application programs they interact with the database system

Sophisticated users :

These user interact with the database system without writing programs, But they submit queries to retrieve the information

Specialized users:

Who write specialized database applications to interact with the database system.

Naïve users:

Interacts with the database system by invoking some application programs that have been written previously by application programmers

Eg : people accessing database over the web

Database Administrator:

Coordinates all the activities of the database system; the database administrator has a good understanding of the enterprise‘s information resources and needs.

- Schema definition
- Access method definition
- Schema and physical organization modification
- Granting user authority to access the database
- Monitoring performance

Storage Manager

The Storage Manager include these following components/modules

- Authorization Manager
- Transaction Manager
- File Manager
- Buffer Manager

- Storage manager is a program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.
- **The storage manager is responsible to the following tasks:**
  - interaction with the file manager
  - efficient storing, retrieving and updating of data

Authorization Manager

- Checks whether the user is an authorized person or not
- Test the satisfaction of integrity constraints

Transaction Manager

Responsible for concurrent transaction execution It ensures that the database remains in a consistent state despite of the system failure

EVOLUTION OF RDBMS

Before the acceptance of Codd's Relational Model, database management systems was just an ad hoc collection of data designed to solve a particular type of problem, later extended to solve more basic purposes. This led to complex systems, which were difficult to understand, install, maintain and use. These database systems were plagued with the following problems:

- They required large budgets and staffs of people with special skills that were in short supply.
- Database administrators' staff and application developers required prior preparation to access these database systems.
- End-user access to the data was rarely provided.
- These database systems did not support the implementation of business logic as a DBMS responsibility.

Hence, the objective of developing a relational model was to address each and every one of the shortcomings that plagued those systems that existed at the end of the 1960s decade, and make DBMS products more widely appealing to all kinds of users.

The existing relational database management systems offer powerful, yet simple solutions for a wide variety of commercial and scientific application problems. Almost every industry uses relational systems to store, update and retrieve data for operational, transaction, as well as decision support systems.

RELATIONAL DATABASE

A relational database is a database system in which the database is organized and accessed according to the relationships between data items without the need for any consideration of physical orientation and relationship. Relationships between data items are expressed by means of **tables**.

It is a tool, which can help you store, manage and disseminate information of various kinds. It is a collection of objects, tables, queries, forms, reports, and macros, all stored in a computer program all of which are inter-related.

It is a method of structuring data in the form of records, so that relations between different entities and attributes can be used for data access and transformation.

RELATIONAL DATABASE MANAGEMENT SYSTEM

A Relational Database Management System (RDBMS) is a system, which allows us to perceive data as tables (and nothing but tables), and ***operators*** necessary to manipulate that data are at the user's disposal.

Features of an RDBMS

The features of a relational database are as follows:

- The ability to create multiple relations (tables) and enter data into them
- An interactive query language
- Retrieval of information stored in more than one table
- Provides a *Catalog* or *Dictionary*, which itself consists of tables (called *system* tables)

Basic Relational Database Terminology

Catalog:

A catalog consists of all the information of the various schemas (external, conceptual and internal) and also all of the corresponding mappings (external/conceptual, conceptual/internal).

It contains detailed information regarding the various objects that are of interest to the system itself; e.g., tables, views, indexes, users, integrity rules, security rules, etc.

In a relational database, the entities of the ERD are represented as *tables* and their attributes as the *columns* of their respective tables in a database schema.

It includes some important terms, such as:

- *Table*: Tables are the basic storage structures of a database where data about something in the real world is

stored. It is also called a *relation or an entity.*

- *Row:* Rows represent collection of data required for a particular entity. In order to identify each row as unique there should be a *unique identifier* called the *primary key*, which allows no duplicate rows. For example in a library every member is unique and hence is given a membership number, which uniquely identifies each member. A row is also called a *record* or a *tuple.*
- *Column:* Columns represent characteristics or attributes of an entity. Each attribute maps onto a column of a table. Hence, a column is also known as an *attribute*.
- *Relationship:* Relationships represent a logical link between two tables. A relationship is depicted by a *foreign key* column.
- Degree: number of attributes
- Cardinality: number of tuples
- An attribute of an entity has a particular value. The set of possible values That a given attribute can have is called its *domain*.

KEYS AND THEIR USE

Key: An attribute or set of attributes whose values uniquely identify each entity in an entity set is called a key for that entity set.

Super Key: If we add additional attributes to a key, the resulting combination would still uniquely identify an instance of the entity set. Such augmented keys are called super keys. **Primary Key:** It is a minimum super key.

It is *a unique* ***identifier*** *for the table* (a column or a column combination with the property that at any given time no two rows of the table contain the same value in that column or column combination).

Foreign Key: A foreign key is a field (or collection of fields) in one table that uniquely identifies a row of another table. In simpler words, the foreign key is defined in a second table, but it refers to the primary key in the first table. **Candidate Key:** There may be two or more attributes or combinations of attributes that uniquely identify an instance of an entity set. These attributes or combinations of attributes are called candidate keys.

Secondary Key: A secondary key is an attribute or combination of attributes that may not be a candidate key, but that classifies the entity set on a particular characteristic. Any key consisting of a single attribute is called a **simple key,** while that consisting of a combination of attributes is called a **composite key**.

Referential Integrity

Referential Integrity can be defined as an integrity constraint that specifies that the value (or existence) of an attribute in one relation depend on the value (or existence) of an attribute in the same or another relation. Referential integrity in a relational database is consistency between coupled tables. It is usually enforced by the combination of a primary key and a foreign key. For referential integrity to hold, any field in a table that is declared a foreign key can contain only values from a parent table's primary key field. For instance, deleting a record that contains a value referred to by a foreign key in another table would break referential integrity.

Relational Model

Relational data model is the primary data model, which is used widely around the world for data storage and processing. This model is simple and it has all the properties and capabilities required to process data with storage efficiency.

Concepts

Tables − In relational data model, relations are saved in the format of Tables. This format stores the relation among entities. A table has rows and columns, where rows represents records and columns represent the attributes.

Tuple − A single row of a table, which contains a single record for that relation is called a tuple.

Relation instance − A finite set of tuples in the relational database system represents relation instance. Relation instances do not have duplicate tuples.

Relation schema − A relation schema describes the relation name (table name), attributes, and their names.

Relation key − Each row has one or more attributes, known as relation key, which can identify the row in the relation (table) uniquely.

Attribute domain − Every attribute has some pre-defined value scope, known as attribute domain.

Constraints

Every relation has some conditions that must hold for it to be a valid relation. These conditions are called **Relational Integrity Constraints**. There are three main integrity constraints −

- Key constraints
- Domain constraints
- Referential integrity constraints

Key Constraints

There must be at least one minimal subset of attributes in the relation, which can identify a tuple uniquely. This minimal subset of attributes is called **key**for that relation. If there are more than one such minimal subsets, these are called ***candidate keys***.

Key constraints force that −

- in a relation with a key attribute, no two tuples can have identical values for key attributes.
- a key attribute can not have NULL values.

Key constraints are also referred to as Entity Constraints.

Domain Constraints

Attributes have specific values in real-world scenario. For example, age can only be a positive integer. The same constraints have been tried to employ on the attributes of a relation. Every attribute is bound to have a specific range of values. For example, age cannot be less than zero and telephone numbers cannot contain a digit outside 0-

Referential integrity Constraints

Referential integrity constraints work on the concept of Foreign Keys. A foreign key is a key attribute of a relation that can be referred in other relation.

Referential integrity constraint states that if a relation refers to a key attribute of a different or same relation, then that key element must exist.

Relational database systems are expected to be equipped with a query language that can assist its users to query the database instances. There are two kinds of query languages − relational algebra and relational calculus.

Relational Algebra

Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either **unary** or **binary**. They accept relations as their input and yield relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

The fundamental operations of relational algebra are as follows −

- Select
- Project
- Union
- Set different
- Cartesian product
- Rename

We will discuss all these operations in the following sections.

Select Operation (σ)

It selects tuples that satisfy the given predicate from a relation.

Notation − $\sigma_p(r)$

Where **σ** stands for selection predicate and **r** stands for relation. *p* is prepositional logic formula which may use connectors like **and, or,** and **not**. These terms may use relational operators like − =, ≠, ≥, < , >, ≤.

For example −

$\sigma_{subject = "database"}(Books)$

Output − Selects tuples from books where subject is 'database'.

$\sigma_{subject = "database" \text{ and } price = "450"}(Books)$

Output − Selects tuples from books where subject is 'database' and 'price' is 450.

$\sigma_{subject = "database" \text{ and } price = "450" \text{ or } year > "2010"}(Books)$

Output − Selects tuples from books where subject is 'database' and 'price' is 450 or those books published after 2010.

Project Operation (∏)

It projects column(s) that satisfy a given predicate.

Notation − $\prod_{A1, A2, An}(r)$

Where A_1, A_2 , A_n are attribute names of relation **r**.

Duplicate rows are automatically eliminated, as relation is a set.

For example −

$\prod_{subject, author}(Books)$

Selects and projects columns named as subject and author from the relation Books.

It performs binary union between two given relations and is defined as −

Notation − r U s

Where **r** and **s** are either database relations or relation result set (temporary relation).

For a union operation to be valid, the following conditions must hold −

- **r**, and **s** must have the same number of attributes.
- Attribute domains must be compatible.

- Duplicate tuples are automatically eliminated.

Output − Projects the names of the authors who have either written a book or an article or both.

Set Difference (−)

The result of set difference operation is tuples, which are present in one relation but are not in the second relation.

Notation − **r** − **s**

Finds all the tuples that are present in **r** but not in **s**.

$\prod_{author}(Books) - \prod_{author}(Articles)$

Output − Provides the name of authors who have written books but not articles.

Cartesian Product (X)

Combines information of two different relations into one.

Notation − r X s

$\sigma_{author = 'tutorialspoint'}(Books\ X\ Articles)$

Output − Yields a relation, which shows all the books and articles written by tutorialspoint.

Rename Operation (ρ)

The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter **rho** *ρ*.

Notation − $\rho_x(E)$

Where the result of expression **E** is saved with name of **x**.

Additional operations are −

- Set intersection
- Assignment
- Natural join

SQL FUNDAMENTALS:

SQL is a standard computer language for accessing and manipulating databases.

What is SQL?

- SQL stands for **S**tructured **Q**uery **L**anguage
- SQL allows you to access a database
- SQL is an ANSI standard computer language
- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert new records in a database
- SQL can delete records from a database
- SQL can update records in a database
- SQL is easy to learn

SQL is a Standard - BUT....

SQL is an ANSI (American National Standards Institute) standard computer language for accessing and manipulating database systems. SQL statements are used to retrieve and update data in a database. SQL works with database programs like MS Access, DB2, Informix, MS SQL Server, Oracle, Sybase, etc.Unfortunately, there are many different versions of the SQL language, but to be in compliance with the ANSI standard, they must support the same major keywords in a similar manner (such as SELECT, UPDATE, DELETE, INSERT, WHERE, and others).

Note: Most of the SQL database programs also have their own proprietary extensions in addition to the SQL standard!

SQL Database Tables

A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"). Tables contain records (rows) with data.Below is an example of a table called "Persons":

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

The table above contains three records (one for each person) and four columns (LastName, FirstName, Address, and City).

SQL Queries

With SQL, we can query a database and have a result set returned.

A query like this:

```
SELECT LastName FROM Persons
```

Gives a result set like this:

LastName
Hansen
Svendson
Pettersen

Note: Some database systems require a semicolon at the end of the SQL statement. We don't use the semicolon in our tutorials.

SQL Data Manipulation Language (DML)

SQL (Structured Query Language) is a syntax for executing queries. But the SQL language also includes a syntax to update, insert, and delete records.

These query and update commands together form the Data Manipulation Language (DML) part of SQL:

- **SELECT** - extracts data from a database table
- **UPDATE** - updates data in a database table
- **DELETE** - deletes data from a database table
- **INSERT INTO** - inserts new data into a database table

SQL Data Definition Language (DDL)

The Data Definition Language (DDL) part of SQL permits database tables to be created or deleted. We can also define indexes (keys), specify links between tables, and impose constraints between database tables. The most important DDL statements in SQL are:

- **CREATE TABLE** - creates a new database table
- **ALTER TABLE** - alters (changes) a database table
- **DROP TABLE** - deletes a database table
- **CREATE INDEX** - creates an index (search key)
- **DROP INDEX** - deletes an index

The SQL SELECT Statement

The SELECT statement is used to select data from a table. The tabular result is stored in a result table (called the result-set).

Syntax

```
SELECT column_name(s)
FROM table_name
```

Note: SQL statements are not case sensitive. SELECT is the same as select.

SQL SELECT Example

To select the content of columns named "LastName" and "FirstName", from the database table called "Persons", use a SELECT statement like this:

```
SELECT LastName,FirstName FROM Persons
```

The database table "Persons":

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

The result

LastName	FirstName
Hansen	Ola
Svendson	Tove
Pettersen	Kari

Select All Columns

To select all columns from the "Persons" table, use a * symbol instead of column names, like this:

```
SELECT * FROM Persons
```

Result

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

The Result Set

The result from a SQL query is stored in a result-set. Most database software systems allow navigation of the result set with programming functions, like: Move-To-First-Record, Get-Record-Content, Move-To-Next-Record, etc. Programming functions like these are not a part of this tutorial. To learn about accessing data with function calls,

Semicolon after SQL Statements?

Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.

Some SQL tutorials end each SQL statement with a semicolon. Is this necessary? We are using MS Access and SQL Server 2000 and we do not have to put a semicolon after each SQL statement, but some database programs force you to use it.

The SELECT DISTINCT Statement

The DISTINCT keyword is used to return only distinct (different) values.

The SELECT statement returns information from table columns. But what if we only want to select distinct elements?

With SQL, all we need to do is to add a DISTINCT keyword to the SELECT statement:

Syntax

```
SELECT DISTINCT column_name(s)
FROM table_name
```

Using the DISTINCT keyword

To select ALL values from the column named "Company" we use a SELECT statement like this:

```
SELECT Company FROM Orders
```

"Orders" table

Company	OrderNumber
Sega	3412
W3Schools	2312
Trio	4678
W3Schools	6798

Result

Company
Sega
W3Schools
Trio
W3Schools

Note that "W3Schools" is listed twice in the result-set.

To select only DIFFERENT values from the column named "Company" we use a SELECT DISTINCT statement like this:

```
SELECT DISTINCT Company FROM Orders
```

Result:

Company
Sega
W3Schools
Trio

Now "W3Schools" is listed only once in the result-set.

The WHERE clause is used to specify a selection criterion.

The WHERE Clause

To conditionally select data from a table, a WHERE clause can be added to the SELECT statement.

Syntax

```
SELECT column FROM table
WHERE column operator value
```

With the WHERE clause, the following operators can be used:

Operator	Description
=	Equal
<>	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern

Note: In some versions of SQL the <> operator may be written as !=

Using the WHERE Clause

To select only the persons living in the city "Sandnes", we add a WHERE clause to the SELECT statement:

```
SELECT * FROM Persons
WHERE City='Sandnes'
```

"Persons" table

LastName	FirstName	Address	City	Year
Hansen	Ola	Timoteivn 10	Sandnes	1951
Svendson	Tove	Borgvn 23	Sandnes	1978
Svendson	Stale	Kaivn 18	Sandnes	1980
Pettersen	Kari	Storgt 20	Stavanger	1960

Result

LastName	FirstName	Address	City	Year
Hansen	Ola	Timoteivn 10	Sandnes	1951
Svendson	Tove	Borgvn 23	Sandnes	1978
Svendson	Stale	Kaivn 18	Sandnes	1980

Using Quotes

Note that we have used single quotes around the conditional values in the examples.
SQL uses single quotes around text values (most database systems will also accept double quotes). Numeric values should not be enclosed in quotes.
For text values:

```
This is correct:
SELECT * FROM Persons WHERE FirstName='Tove'
This is wrong:
SELECT * FROM Persons WHERE FirstName=Tove
```

For numeric values:

```
This is correct:
SELECT * FROM Persons WHERE Year>1965
This is wrong:
SELECT * FROM Persons WHERE Year>'1965'
```

The LIKE Condition

The LIKE condition is used to specify a search for a pattern in a column.

Syntax

```
SELECT column FROM table
WHERE column LIKE pattern
```

A "%" sign can be used to define wildcards (missing letters in the pattern) both before and after the pattern.

Using LIKE

The following SQL statement will return persons with first names that start with an 'O':

```
SELECT * FROM Persons
WHERE FirstName LIKE 'O%'
```

The following SQL statement will return persons with first names that end with an 'a':

```
SELECT * FROM Persons
WHERE FirstName LIKE '%a'
```

The following SQL statement will return persons with first names that contain the pattern 'la':

```
SELECT * FROM Persons
WHERE FirstName LIKE '%la%'
```

The INSERT INTO Statement

The INSERT INTO statement is used to insert new rows into a table.

Syntax

```
INSERT INTO table_name
VALUES (value1, value2,....)
```

You can also specify the columns for which you want to insert data:

```
INSERT INTO table_name (column1, column2,...)
VALUES (value1, value2,....)
```

Insert a New Row

This "Persons" table:

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger

And this SQL statement:

```
INSERT INTO Persons
VALUES ('Hetland', 'Camilla', 'Hagabakka 24', 'Sandnes')
```

Will give this result:

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Hetland	Camilla	Hagabakka 24	Sandnes

Insert Data in Specified Columns

This "Persons" table:

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Hetland	Camilla	Hagabakka 24	Sandnes

And This SQL statement:

```
INSERT INTO Persons (LastName, Address)
VALUES ('Rasmussen', 'Storgt 67')
```

Will give this result:

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Hetland	Camilla	Hagabakka 24	Sandnes
Rasmussen		Storgt 67	

Null (no value …not space not empty)

The UPDATE statement is used to modify the data in a table.

Syntax

```
UPDATE table_name
SET column_name = new_value
WHERE column_name = some_value
```

Person:

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen		Storgt 67	

Update one Column in a Row

We want to add a first name to the person with a last name of ―Rasmussen‖:

```
UPDATE Person SET FirstName = _Nina‘
WHERE LastName = _Rasmussen‘
```

Result:

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Storgt 67	

Update several Columns in a Row

We want to change the address and add the name of the city:

```
UPDATE Person
SET Address = 'Stien 12', City = 'Stavanger'
WHERE LastName = 'Rasmussen'
```

Result:

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Stien 12	Stavanger

The DELETE Statement

The DELETE statement is used to delete rows in a table.

Syntax

```
DELETE FROM table_name
WHERE column_name = some_value
```

Person:

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Stien 12	Stavanger

Delete
Drop

Delete a Row

"Nina Rasmussen" is going to be deleted:

```
DELETE FROM Person WHERE LastName = 'Rasmussen'
```

Result

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger

Delete All Rows

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

```
DELETE FROM table_name
or
DELETE * FROM table_name
```

The ORDER BY keyword is used to sort the result.

Sort the Rows

The ORDER BY clause is used to sort the rows.

Orders:

Company	OrderNumber
Sega	3412
ABC Shop	5678
W3Schools	2312
W3Schools	6798

Example

To display the company names in alphabetical order:

```
SELECT Company, OrderNumber FROM Orders
ORDER BY Company ASC (asending)
```

Result:

Company	OrderNumber
ABC Shop	5678
Sega	3412
W3Schools	6798
W3Schools	2312

Example

To display the company names in alphabetical order AND the OrderNumber in numerical order:

```
SELECT Company, OrderNumber FROM Orders
ORDER BY Company, OrderNumber
```

Result:

Company	OrderNumber
ABC Shop	5678
Sega	3412
W3Schools	2312
W3Schools	6798

Aggregate functions

Aggregate functions operate against a collection of values, but return a single value.

Note: If used among many other expressions in the item list of a SELECT statement, the SELECT must have a GROUP BY clause!!

"Persons" table (used in most examples)

Name	Age
Hansen, Ola	34
Svendson, Tove	45
Pettersen, Kari	19

Aggregate functions in MS Access

Function	Description
AVG(column)	Returns the average value of a column
COUNT(column)	Returns the number of rows (without a NULL value) of a column
COUNT(*)	Returns the number of selected rows
FIRST(column)	Returns the value of the first record in a specified field
LAST(column)	Returns the value of the last record in a specified field
MAX(column)	Returns the highest value of a column
MIN(column)	Returns the lowest value of a column
STDEV(column)	
STDEVP(column)	
SUM(column)	Returns the total sum of a column
VAR(column)	

VARP(column)	

Aggregate functions in SQL Server

Function	Description
AVG(column)	Returns the average value of a column
BINARY_CHECKSUM	
CHECKSUM	
CHECKSUM_AGG	
COUNT(column)	Returns the number of rows (without a NULL value) of a column
COUNT(*)	Returns the number of selected rows
COUNT(DISTINCT column)	Returns the number of distinct results
FIRST(column)	Returns the value of the first record in a specified field (not supported in SQLServer2K)
LAST(column)	Returns the value of the last record in a specified field (not supported in SQLServer2K)
MAX(column)	Returns the highest value of a column
MIN(column)	Returns the lowest value of a column
STDEV(column)	
STDEVP(column)	
SUM(column)	Returns the total sum of a column
VAR(column)	
VARP(column)	

Scalar functions

Scalar functions operate against a single value, and return a single value based on the input value.

Useful Scalar Functions in MS Access

Function	Description
UCASE(c)	Converts a field to upper case
LCASE(c)	Converts a field to lower case
MID(c,start[,end])	Extract characters from a text field
LEN(c)	Returns the length of a text field
INSTR(c,char)	Returns the numeric position of a named character within a text field
LEFT(c,number_of_char)	Return the left part of a text field requested
RIGHT(c,number_of_char)	Return the right part of a text field requested
ROUND(c,decimals)	Rounds a numeric field to the number of decimals specified
MOD(x,y)	Returns the remainder of a division operation

Aggregate functions (like SUM) often need an added GROUP BY functionality.

GROUP BY

GROUP BY... was added to SQL because aggregate functions (like SUM) return the aggregate of all column values every time they are called, and without the GROUP BY function it was impossible to find the sum for each individual group of column values.

The syntax for the GROUP BY function is:

```
SELECT column,SUM(column) FROM table GROUP BY column
```

GROUP BY Example

This "Sales" Table:

Company	Amount
W3Schools	5500
IBM	4500
W3Schools	7100

And This SQL:

```
SELECT Company, SUM(Amount) FROM Sales
```

Returns this result:

Company	SUM(Amount)
W3Schools	17100
IBM	17100
W3Schools	17100

The above code is invalid because the column returned is not part of an aggregate. A GROUP BY clause will solve this problem:

```
SELECT Company,SUM(Amount) FROM Sales
GROUP BY Company
```

Returns this result:

Company	SUM(Amount)
W3Schools	12600
IBM	4500

HAVING…

HAVING... was added to SQL because the WHERE keyword could not be used against aggregate functions (like SUM), and without HAVING... it would be impossible to test for result conditions.

The syntax for the HAVING function is:

```
SELECT column,SUM(column) FROM table
GROUP BY column
HAVING SUM(column) condition value
```

This "Sales" Table:

Company	Amount
W3Schools	5500
IBM	4500
W3Schools	7100

This SQL:

```
SELECT Company,SUM(Amount) FROM Sales
GROUP BY Company
HAVING SUM(Amount)>10000
```

Returns this result

Company	SUM(Amount)
W3Schools	12600

EMBEDDED SQL

Embedded SQL is a method of inserting inline SQL statements or queries into the code of a programming language, which is known as a host language. Because the host language cannot parse SQL, the inserted SQL is parsed by an embedded SQL preprocessor.

Embedded SQL is a robust and convenient method of combining the computing power of a programming language with SQL's specialized data management and manipulation capabilities.

Structure of embedded SQL

Structure of embedded SQL defines step by step process of establishing a connection with DB and executing the code in the DB within the high level language.

Connection to DB

This is the first step while writing a query in high level languages. First connection to the DB that we are accessing needs to be established. This can be done using the keyword CONNECT. But it has to precede with _EXEC SQL' to

indicate that it is a SQL statement.
EXEC SQL CONNECT db_name;
EXEC SQL CONNECT HR_USER; //connects to DB HR_USER

Once connection is established with DB, we can perform DB transactions. Since these DB transactions are dependent on the values and variables of the host language. Depending on their values, query will be written and executed. Similarly, results of DB query will be returned to the host language which will be captured by the variables of host language. Hence we need to declare the variables to pass the value to the query and get the values from query. There are two types of variables used in the host language.

- **Host variable :** These are the variables of host language used to pass the value to the query as well as to capture the values returned by the query. Since SQL is dependent on host language we have to use variables of host language and such variables are known as host variable. But these host variables should be declared within the SQL area or within SQL code. That means compiler should be able to differentiate it from normal C variables. Hence we have to declare host variables within BEGIN DECLARE and END DECLARE section. Again, these declare block should be enclosed within EXEC SQL and _;‘.

```
EXEC SQL BEGIN DECLARE SECTION;
        int STD_ID;
        char STD_NAME [15];
        char ADDRESS[20];
EXEC SQL END DECLARE SECTION;
```

We can note here that variables are written inside begin and end block of the SQL, but they are declared using C code. It does not use SQL code to declare the variables. Why? This is because they are host variables – variables of C language. Hence we cannot use SQL syntax to declare them. Host language supports almost all the datatypes from int, char, long, float, double, pointer, array, string, structures etc.

When host variables are used in a SQL query, it should be preceded by colon – _:‘ to indicate that it is a host variable. Hence when pre-compiler compiles SQL code, it substitutes the value of host variable and compiles. EXEC SQL SELECT * FROM STUDENT WHERE STUDENT_ID =:STD_ID;

The following code is a simple embedded SQL program, written in C. The program illustrates many, but not all, of the embedded SQL techniques. The program prompts the user for an order number, retrieves the customer number, salesperson, and status of the order, and displays the retrieved information on the screen.

```
int main() {
  EXEC SQL INCLUDE SQLCA;
  EXEC SQL BEGIN DECLARE SECTION;
    int OrderID;        /* Employee ID (from user)         */
    int CustID;         /* Retrieved customer ID           */
    char SalesPerson[10]  /* Retrieved salesperson name    */
    char Status[6]      /* Retrieved order status          */
  EXEC SQL END DECLARE SECTION;

  /* Set up error processing */
  EXEC SQL WHENEVER SQLERROR GOTO query_error;
  EXEC SQL WHENEVER NOT FOUND GOTO bad_number;

  /* Prompt the user for order number */
  printf ("Enter order number: ");
  scanf_s("%d", &OrderID);

  /* Execute the SQL query */
  EXEC SQL SELECT CustID, SalesPerson, Status
    FROM Orders
    WHERE OrderID = :OrderID
    INTO :CustID, :SalesPerson, :Status;
```

```
/* Display the results */
  printf ("Customer number: %d\n", CustID);
  printf ("Salesperson: %s\n", SalesPerson);
  printf ("Status: %s\n", Status);
  exit();

query_error:
  printf ("SQL error: %ld\n", sqlca->sqlcode);
  exit();

bad_number:
  printf ("Invalid order number.\n");
  exit();
}
```

DYNAMIC SQL

The main disadvantage of embedded SQL is that it supports only static SQLs. If we need to build up queries at run time, then we can use dynamic sql. That means if query changes according to user input, then it always better to use dynamic SQL. Like we said above, the query when user enters student name alone and when user enters both student name and address, is different. If we use embedded SQL, one cannot implement this requirement in the code. In such case dynamic SQL helps the user to develop query depending on the values entered by him, without making him know which query is being executed. It can also be used when we do not know which SQL statements like Insert, Delete update or select needs to be used, when number of host variables is unknown, or when datatypes of host variables are unknown or when there is direct reference to DB objects like tables, views, indexes are required.

However this will make user requirement simple and easy but it may make query lengthier and complex. That means depending upon user inputs, the query may grow or shrink making the code flexible enough to handle all the possibilities. In embedded SQL, compiler knows the query in advance and pre-compiler compiles the SQL code much before C compiles the code for execution. Hence embedded SQLs will be faster in execution. But in the case of dynamic SQL, queries are created, compiled and executed only at the run time. This makes the dynamic SQL little complex, and time consuming.

Since query needs to be prepared at run time, in addition to the structures discussed in embedded SQL, we have three more clauses in dynamic SQL. These are mainly used to build the query and execute them at run time.

PREPARE

Since dynamic SQL builds a query at run time, as a first step we need to capture all the inputs from the user. It will be stored in a string variable. Depending on the inputs received from the user, string variable is appended with inputs and SQL keywords. These SQL like string statements are then converted into SQL query. This is done by using PREPARE statement.

For example, below is the small snippet from dynamic SQL. Here sql_stmt is a character variable, which holds inputs from the users along with SQL commands. But is cannot be considered as SQL query as it is still a sting value. It needs to be converted into a proper SQL query which is done at the last line using PREPARE statement. Here sql_query is also a string variable, but it holds the string as a SQL query.

EXECUTE

This statement is used to compile and execute the SQL statements prepared in DB.

EXEC SQL EXECUTE sql_query;

EXECUTE IMMEDIATE

This statement is used to prepare SQL statement as well as execute the SQL statements in DB. It performs the task of PREPARE and EXECUTE in a single line.

EXEC SQL EXECUTE IMMEDIATE :sql_stmt;

Dynamic SQL will not have any SELECT queries and host variables. But it can be any other SQL statements like insert, delete, update, grant etc. But when we use insert/ delete/ updates in this type, we cannot use host variables. All the input values will be hardcoded. Hence the SQL statements can be directly executed using EXECUTE IMMEDIATE rather than using PREPARE and then EXECUTE.

```
EXEC SQL EXECUTE IMMEDIATE ‗GRANT SELECT ON STUDENT TO Faculty‘;
EXEC SQL EXECUTE IMMEDIATE ‗DELETE FROM STUDENT WHERE STD_ID = 100‘;
EXEC SQL EXECUTE IMMEDIATE ‗UPDATE STUDENT SET ADDRESS = ‗Troy‘ WHERE STD_ID =100‘;
```