# DATABASE FUNDAMENTALS CONCEPTS

The term **database** refers to a collection of related data from which the users can efficiently retrieve the desired information. In addition to the storage and retrieval of data, certain other operations can also be performed on a database. These operations include adding, updating and deleting data. All these operations on a database are performed using a database management system (DBMS). Essentially, a DBMS is a computerized record-keeping system. In this topic we will be introduced to the basic terminology used in a database management system (such as ***normalisation, entities, attributes, keys, relational database management systems, structured query language***).

## Table of Contents

## I. DATA, INFORMATION AND KNOWLEDGE

### I.1 Definitions

Data can be anything such as a number, a person's name, images, sounds and so on. Hence, data can be defined as a set of isolated and unrelated raw facts (represented by values), which have little or no meaning because they lack a context for evaluation (e.g. 'Monica', '36', 'chief' …). When the data are processed and converted into a meaningful and useful form, it is known as information. Hence, information can be defined as a set of organized and validated collection of data. For example, *'Monica is 35 years old and she is a chef'*.

Strictly speaking, data refer to the values physically recorded in the database, whereas information refers to the conclusion or meaning drawn out of it. With respect to database, these terms are synonymous.

Other than data and information, one more term, knowledge, is frequently used with database technology. Knowledge is the act of understanding the context in which the information is used. It can be based on learning through information, experience and/or intuition.

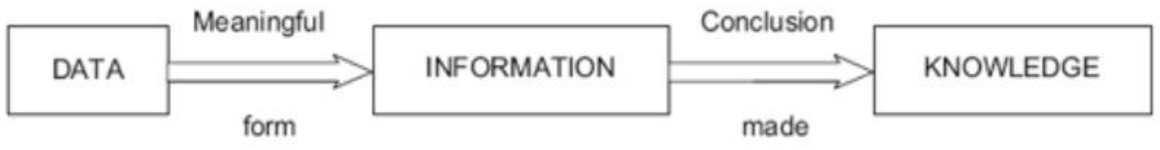


*Figure 1. Data, Information and Knowledge*

**Data Model:** A **data model** is a representation of a real world situation about which data is to be collected and stored in a database. A data model depicts the dataflow and logical interrelationships among different data elements.

### I.2 Prerequisites of Information

Information is the processed data, on which decisions are taken and the subsequent actions are performed thereafter. For the decisions to be meaningful and useful, the information must possess the following qualities:

- **Accurate:** To be useful, information must be accurate at all levels because all further developments are based on the available information.
- **Timely:** Information is appreciated only if it is available on time.
- **Complete:** Complete information tends to be comprehensive in covering the issue or topic of interest.
- **Precise:** Information should be to the point, containing all the essential elements of the relevant subject areas.
- **Relevant:** Information is relevant if it can be applied to a specific situation, problem or issue of interest.

### I.3 Need for Information

Information is an important part of our day-to-day life. Almost all activities are affected by the quantity as well as the quality of information. Some of the common usages of information are discussed as follows.

#### *I.3.1 Information and Decision-making:*

Decision-making is the process of identifying, selecting and implementing the best possible alternative. The right information, in the right form and at the right time is essential to make correct decisions.

#### *I.3.2 Information and Communication:*

Information is vital for communication and is a critical resource for performing work in organizations. Business managers spend most of their day in communicating with other managers, subordinates, customers, vendors and so on. A manager must keep track of the information flow from the sources inside and outside the organization.

#### *I.3.3 Information and Knowledge:*

The future is shaped by our actions today, and these actions are based upon our knowledge. Therefore, for achieving higher levels of success, one must be well informed and should have clarity of information.

#### *I.3.4 Information and Productivity:*

Information helps in making sense of our environment, which assists in achieving the performance objectives. In fact, productivity is directly related to the availability and value of the information and its application in the related context.

## II. DATABASE: DEFINITION

A **database** is a collection of non-redundant data which can be shared by different application systems. Although databases are generally **computerized**, instances of **non-computerized databases** from everyday life can be cited in abundance. A dictionary, a phone book, a collection of recipes and a TV guide are all common examples of non-computerized databases. The examples of computerized databases include customer files, employee rosters, books catalogue, equipment inventories and sales transactions.

### II.1 Stages in Creating a Database

The process of creating a database can be broadly divided into two main stages:

1. **Data analysis** involves using a formalised methodology to create a database design. Two widely used methods are **Entity Relationship Modelling (ER)** and **Normalisation**.

2. **Physical implementation** of that design in a database system.

## II.2 Database terminologies

Within the database, the data are organized into storage containers, called tables. Tables are made up of columns and rows. In a table, columns represent individual fields and rows represent records of data. The following are the basic database terms.

### *II.2.1 Field*

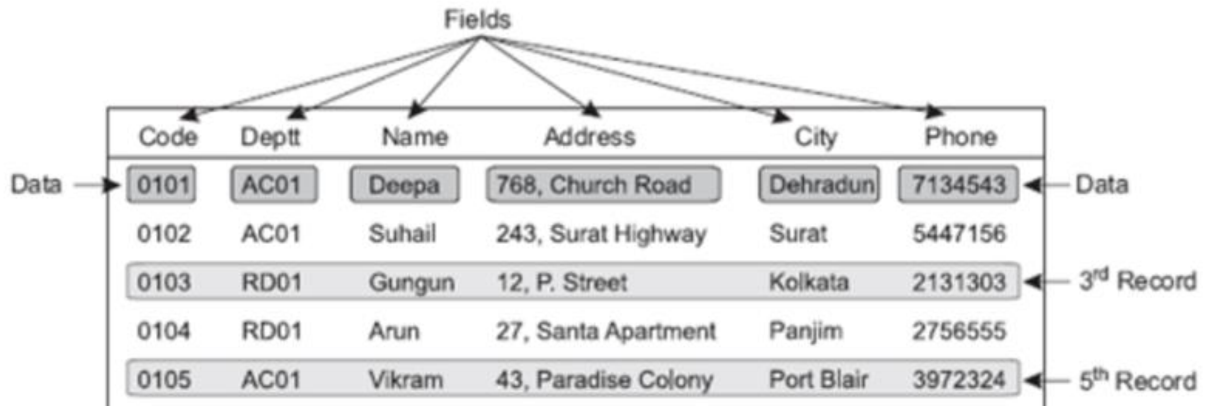
**A field** represents one related part of a table and is the smallest logical structure of storage in a database. It holds one piece of information about an item or a subject. For example, in a database maintaining information about employee, the fields can be Code, Deptt, Name, Address, City and Phone (see Figure 2).

### *II.2.2 Record*

A record is a collection of multiple related fields that can be treated as a unit. For example, fields Code, Deptt, Name, Address, City and Phone for a particular employee form a record. Figure 2 contains nine records (0101–0109) and each record has six fields.

### *II.2.3 Table*

A table is a named collection of logically related multiple records. For example, a collection of all the employee records of a company form employee table. Note that every record in a table has the same set of fields. Depending on the database software, a table can also be referred to as a **file**. The collection of multiple related files (tables) forms the database.


Fields
Data
Data
3rd Record
5th Record

| Code | Deptt | Name | Address | City | Phone |
|---|---|---|---|---|---|
| 0101 | AC01 | Deepa | 768, Church Road | Dehradun | 7134543 |
| 0102 | AC01 | Suhail | 243, Surat Highway | Surat | 5447156 |
| 0103 | RD01 | Gungun | 12, P. Street | Kolkata | 2131303 |
| 0104 | RD01 | Arun | 27, Santa Apartment | Panjim | 2756555 |
| 0105 | AC01 | Vikram | 43, Paradise Colony | Port Blair | 3972324 |

Figure 2. Fields and Records in a Table

### *II.2.4 Data Type*

A **data type** determines the type of data that can be stored in a column. Although many data types are available, the four most commonly used data types are ***Character, Numeric, Boolean*** and ***DateTime***. The values for this data type vary widely depending on the database management software being used.

| Data type | Character | Numeric | Boolean | DateTime |
|---|---|---|---|---|
| Field Name | Name | Salary | Is_Married | Joining_Date |
| Data | Placide | 450000 | False (No) | 02/10/98 |

#### II.1.5 Data Dictionary

Apart from the data, the database also stores metadata, which describes the tables, columns, indexes, constraints and other items that make up the database. In simple words, metadata is the data about data. This metadata is stored in an area called the data dictionary. Hence, a data dictionary defines the basic organization of a database.

Most database systems keep the data dictionary hidden from users to prevent them from accidentally destroying its contents. Different users use the dictionary in different ways.

- **The database administrator** (DBA) needs a dictionary to ensure consistency among the data items, to educate users about the database content and to help ensure that different department defines the same data in the same way.
- **The programmers** may use it to ensure that they have the name and coding of the data items or segments correct in their programs.
- **Managers** may use it as a guide to decide what data could be made available to them.

## III. DATA CONCEPTS

### III.1 Physical Data Concepts

Physical concepts of data refer to the manner in which the data are physically stored on the hardware (like hard disk). Fundamentally, it involves the physical organization of the records of a file for the convenience of storage and retrieval of data. Usually, the files are organized in three fashions: ***sequential***, ***direct*** and ***indexed sequential***.

#### *III.1.1 Sequential Files*

In sequential files, the data are stored and/or retrieved in a logical order, that is, in a sequence. The records are stored one after the other in an ascending or descending order, based on the key field (which is unique for each record) of the records. Generally, these files are stored on sequential storage devices such as magnetic tapes and punched cards. In such files, to retrieve a record, all the records must be traversed sequentially before reaching to the desired record. An analogy to sequential files may be taken as an audio cassette.

#### *III.1.2 Direct Files*

Direct files facilitate accessing any record directly or randomly without having to traverse the sequence of records. These files are also known as random or relative files. Even though only one item can be accessed at a time, that item may be stored anywhere in the file. For example, in case of CDs

#### *III.1.3 Indexed Sequential Files*

Essentially, indexed sequential technique is a hybrid of sequential and direct file organization. The indexed file organization uses a separate index file, which contains the key values and the location of the corresponding record. The records are organized in an orderly sequence and the index table is used to access the records without searching through the entire file.

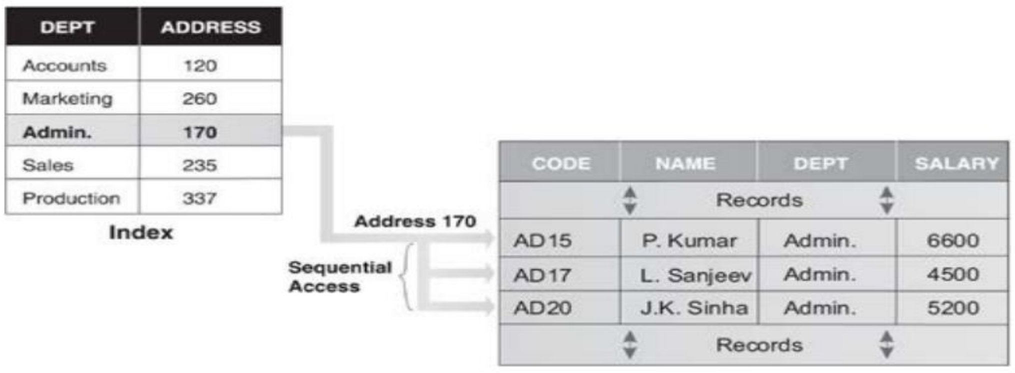


*Figure 3. Indexed Sequential Access*

## III.2 Logical Data Concepts

Once the requirements of the user have been specified, the next step is to construct an **abstract** or **conceptual model** of a database based on the requirements of the user. The **conceptual model** represents various pieces of data and their relationships at a very high level of abstraction. It mainly focuses on what data are required and how it should be organized rather than what operations are to be performed on the data. The conceptual model can be represented using **Entity-Relationship model** (***E-R model***). The E-R model views the real world as a set of basic objects (known as **entities**), their characteristics (known as **attributes**) and associations among these objects (known as **relationships**).

### III.2.1 Entity

An **entity** is any object in the system that we want to model and store information about. Entities are usually recognizable concepts, either concrete or abstract, such as person, places, things, or events which have relevance to the database. Some specific examples of entities are ***Employee, Student, Lecturer***. An entity is analogous to a **table** in the relational model.

An **entity occurrence** is an instance of an entity. For example, in the student entity, the information about each individual student details is an entity occurrence, An entity occurrence can also be referred to as a **record. By** *convention, entities are represented by rectangles:*

### III.2.2 Attributes

An **attribute i**s an item of information which is stored about an entity. For example, the entity 'lecturer' could have attributes such as ***staff id, surname, forename, date of birth, telephone number,*** etc. By convention, an attribute is represented by a **diamond** linked to the corresponding entity:

### III.2.3 Relationship

**Relationship** is an association, dependency or link between two or more entities and is represented by a diamond symbol. A relationship describes how two or more entities are related to each other. For example, the relationship Buys (shown in Figure 4) associates the CUSTOMER entity with ITEMS entity.

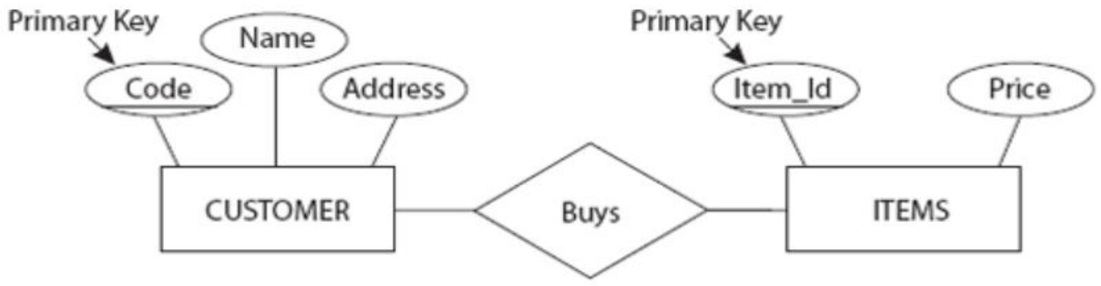


***Figure 4. Entities, Attributes and Relationship***

## III.3 Types of Relationship

Even though a relationship may involve more than two entities, the most commonly encountered relationships are binary, involving exactly two entities. Generally, such binary relationships are of three types and called cardinality: **one-to-one**, **one-to-many** and **many-to-many**.

### *a) One-to-one Relationship (1:1)*

One-to-one is where one occurrence of an entity relates to only one occurrence in another entity, eg if a man only marries one woman and a woman only marries one man, it is a one-to-one (1:1) relationship.

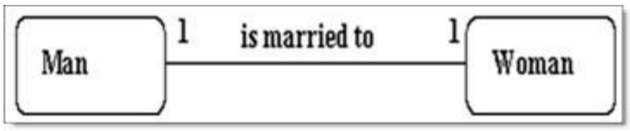


**Fig 5 : One-to-One**

### *b) One-to-many Relationship (1:M)*

A one-to-many relationship is where one occurrence in an entity relates to many occurrences in another entity. For instance one manager manages many employees, but each -employee only has one manager, so it is a one-to many (1:m) relationship.

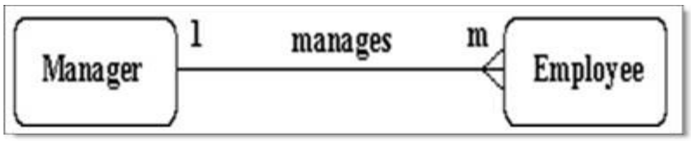
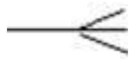


**Fig 6: One-to-Many**

 The crowbar represents the Many occurrence.

### c) Many-to-many Relationship (M:M)

In many-to-many relationship, one record in a table can be related to one or more records in a second table, and one or more records in the second table can be related to one or more records in the first table. For example, One teacher teaches many students and a student is taught by many teachers.

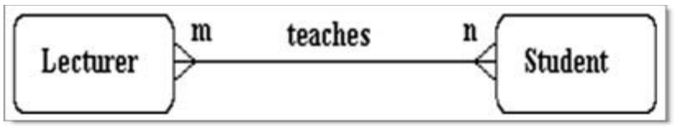


**Fig 7 Many-to-Many relationship**

## III.4 Keys

A **key** is a data item that allows us to uniquely identify individual occurrences or an entity type. You can sort and quickly retrieve information from a database by choosing one or more fields (ie attributes) to act as *keys*. For instance, in a student's table you could use a combination of the last name and first name fields (or perhaps last name, first name and birth dates to ensure you identify each student uniquely) as a key field. There exist many types of keys:

a) ***Primary Key:***

A field or a set of fields that uniquely identify each record in a table is known as a **primary key**. This implies that no two records in the relation can have same value for the primary key. For example, your student number is a primary key as this uniquely identifies you within the college student records system. An employee number uniquely identifies a member of staff within a company. An IP address uniquely addresses a PC on the internet.

A primary key is **mandatory**. That is, each entity occurrence must have a value for its primary key.

*b)* ***Candidate Key:***

In a table, there can be more than one field that can uniquely identify each record. All such fields are known as **candidate keys**. One of these candidate keys is chosen as a primary key; the other keys that are not chosen as primary key are known as **alternate keys or secondary keys**.

*c)* ***Foreign Key:***

A field of a table that references the primary key of another table is referred to as foreign key. Figure 13.3 illustrates how a foreign key constraint is related to a primary key constraint. Here, the field Item_Code in the PURCHASE table references the field Item_Code in the ITEM relation. Thus, the attribute Item_Code in the PURCHASE relation is the foreign key.

***NOTE:*** *The key composed of more than one field is known as* ***composite key****. Sometimes, it is also known as* ***concatenated key*** *or* ***structured key****.*

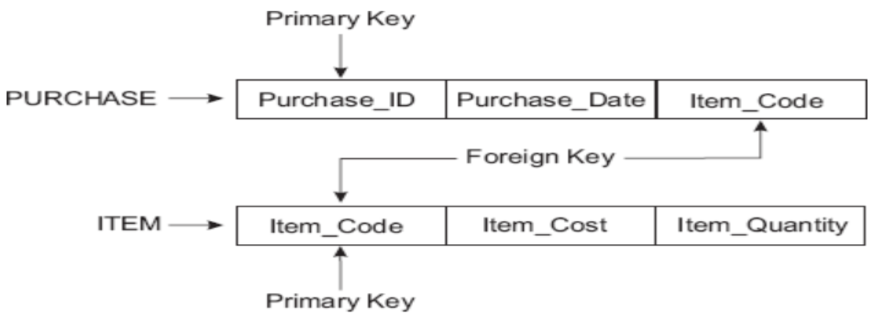


***Figure 8. Foreign Key***

### *d) Simple Key*

Any of the keys described before (ie: primary, secondary or foreign) may have one or more attributes. A **simple key** consists of a **single attribute** to uniquely identify an entity occurrence, for example, a student number, which uniquely identifies a particular student. No two students would have the same student number.

### e) Compound Key

A **compound key** consists of **more than one attribute** to uniquely identify an entity occurrence. Each attribute, which makes up the key, is also a simple key in its own right.

For example, we have an entity named enrolment, which holds the courses on which a student is enrolled. In this scenario a student is allowed to enrol on more than one course. This has a compound key of both student number and course number, which is required to uniquely identify a student on a particular course.

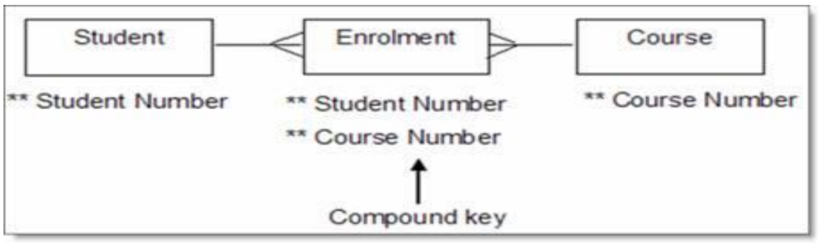


*Fig 9: a compound key*

Student number and course number combined is a compound primary key for the enrolment entity.

### *f) Composite Key*

A **composite key** consists of more than one attribute to uniquely identify an entity occurrence. This differs from a compound key in that one or more of the attributes, which make up the key, are not simple keys in their own right.

For example, you have a database holding your CD collection. One of the entities is called tracks, which holds details of the tracks on a CD. This has a composite key of CD name, track number.

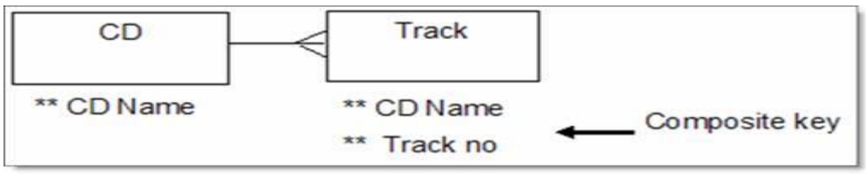


***Fig 10: a composite key***

CD name in the track entity is a simple key, linking to the CD entity, but track number is not a simple key in its own right.

**Application exercise**
For each of the following entities, list possible primary keys. Then, suggest secondary keys, if any: Student, Course, Unit, Result, Classroom, Lecturer, Department, Attendance

## IV. DATA INTEGRITY

**Integrity** ensures that the data in a database is both accurate and complete, in other words, that the data makes sense. There are at least five different types of integrity that need to be considered: ***Domain constraints, Entity integrity, Column constraints, User-defined integrity constraints, Referential integrity***. The data analysis stage will identify the requirements of these.

- ✓ **Domain Constraints:** A domain is defined as the set of all unique values permitted for an attribute. For example, a domain of Date is the set of all possible valid dates, a domain of Integer is all possible whole numbers, and a domain of day-of-week is Monday, Tuesday ... Sunday.
- ✓ **Entity Integrity:** It implies that no component of a primary key is allowed to have a NULL value.
- ✓ **Column Constraints:** During the data analysis phase, business rules will identify any column constraints. For example, a salary cannot be negative; an employee number must be in the range 1000 - 2000, etc.
- ✓ **User-Defined Integrity Constraints:** Business rules may dictate that when a specific action occurs, further actions should be triggered. For example, deletion of a record automatically writes that record to an audit table.
- ✓ **Referential Integrity:** It implies that if a foreign key is defined in one table, any of its value must exist as a primary key in another table.

## V. DATABASE MANAGEMENT SYSTEM

### V.1 Definition

To carry out operations like insertion, deletion and retrieval, the database needs to be managed by a software package. This software is called a **database management system** (***DBMS***). Hence, *DBMS can be defined as a collection of interrelated data and a set of programs to access that data*.

**Database system:** Database system is a general term that refers to the combination of a database, a database management system and a data model. This system is responsible for the following data manipulation acts; data controlling, data retrieving, data maintenance and data definition.

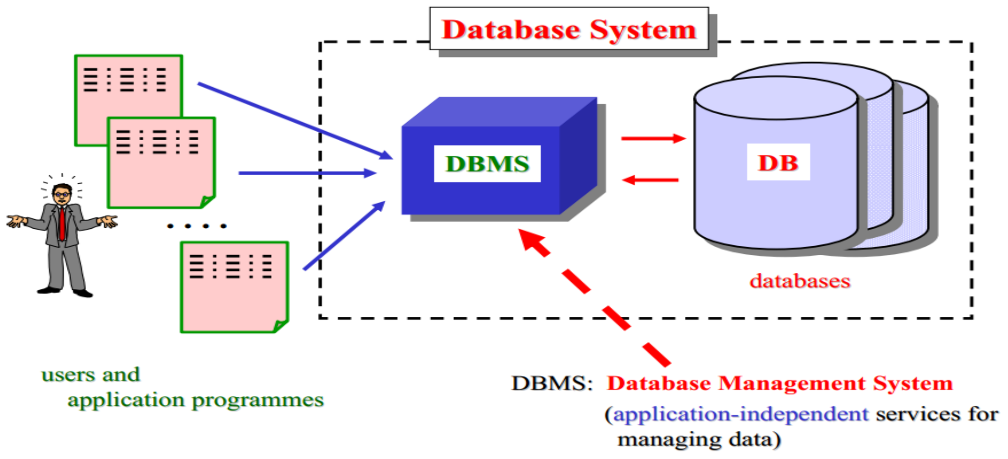

Database System
DBMS
DB
databases
users and application programmes
DBMS: Database Management System
(application-independent services for managing data)

*Figure 11: a database system*

### V.2 Advantages and Limitations of a DBMS

A good database management system (DBMS) should provide the following advantages over a conventional system:

**Advantages**

→ **Reduction in Data Redundancy**: Data redundancy refers to duplication of data. In nondatabase systems, each application has its own separate files. This can often lead to redundancy in stored data, which results in wastage of space.

→ **Reduction in data Inconsistency**: Data inconsistency is when different versions of the same data appear in different places in a database. This causes unreliable information, because it is difficult to determine which version of the information is correct.

→ **Sharing of Data:** Sharing of data allows the existing applications to use the data in the database simultaneously.

→ **Improvement in Data Security:** DBMS can ensure that the only means of accessing the database is through the authorized channel. Hence, data security checks can be carried out whenever access is attempted to sensitive data. To ensure security, DBMS provides security tools such as user codes and passwords.

→ **Maintenance of Data Integrity**: Data integrity means the consistency and accuracy of the data in the database.

→ **Better Interaction with Users**: Centralizing the data in a database also means that users can obtain new and combined information that would have been impossible to obtain otherwise. In addition, use of a DBMS allows the users, who do not know programming, to interact with the data more easily.

However, the following can be viewed as some of the limitations of a database:

**Disadvantages**

→ Database systems are complex, difficult, and time-consuming to design
→ Substantial hardware and software start-up costs
→ Damage to database affects virtually all applications programs
→ Extensive conversion costs in moving form a file-based system to a database system
→ Initial training required for all programmers and users

### V.3 examples of Database Management System (DBMS)

Some of the database management systems are:

1) **Microsoft Access** : This is the database management system developed by Microsoft.
2) **MySQL** : MySQL is open source database management system, one of the most popular dbms on the web. It is reliable, fast and also flexible.
3) **Oracle** : Developed by Oracle corporation. It is object relational database management system. The original version of Oracle software was developed by Software Development Laboratories (SDL). Oracle is regarded to be one of the safe DBMS.
4) **Microsoft SQL Server** : Microsoft developed this relational database server. The primary function of this software is to store and retrieve the data as requested by other applications, whether those applications are on the same computer or running on other computers across the network (including internet).
5) **Filemaker :** Filemaker began as a MS-DOS based computer program named nutshell. It is a cross platform RDBMS developed by Filemaker Inc.

## VI. DATABASE MODELS

**A database model** or simply a data model is an abstract model that describes how the data are organized and represented. A data model consists of two parts, which are as follows:

- **A mathematical notation** for describing the data and relationships
- **A set of operations** used to manipulate that data

Every database and DBMS is based on a particular database model. There are four basic types of database models—hierarchical, network, relational and object-oriented. These models provide different conceptualizations of the database and they have different outlooks and perspectives.

### VI.1 Hierarchical Database Model

The hierarchical data model is the oldest type of data model, developed by IBM in 1968. This data model organizes the data in a tree-like structure in which each child node (also known as dependents) can have only one parent node. In other words, a hierarchal database is a collection of records connected to one another through links. The top of the tree structure consists of a single node that does not have any parent and is called the root node.

The main advantage of the hierarchical data model is that the data access is quite predictable in structure, and therefore, both retrieval and updates can be highly optimized by a DBMS. However, the main drawback of this model is that the links are 'hard coded' into the data structure, that is, the links are permanently established and cannot be modified.

### VI.2 Network Database Model

In a network model, the data are represented by a collection of records and the relationships among data are represented by links. A link is an association between precisely two records.

The main limitation of the network data model is that it can be quite complicated to maintain all the links and a single broken link can lead to problems in the database. In addition, since there are no restrictions on the number of links, the database design can become overwhelmingly complex.

### VI.3 Object-oriented Database Model

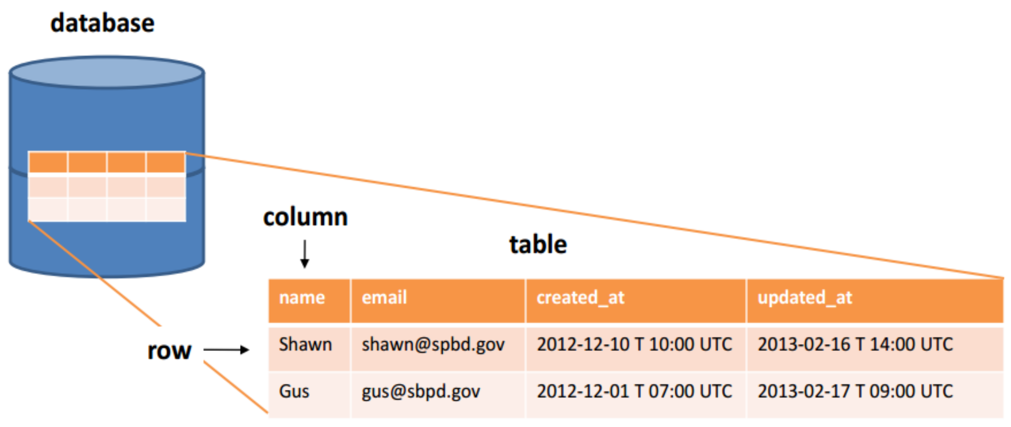
The object-oriented model is a relatively new data model and provides an outlook for the future database models. An object-oriented database stores and maintains objects. An object is an item that can contain both the data and the procedures that manipulate the data. For example, a student object might contain not only data about a student's name, roll number and address, but also procedures on some tasks such as printing the student record or calculating the student's tuition fees.

Like the other models, the object model assumes that objects can conceptually be collected together into meaningful groups known as classes.

### VI.4 Relational Database

**A relational database** is a collection of data items organized as a set of formally-described tables from which data can be accessed or reassembled in many different ways without having to reorganize the database tables. The relational database was invented by E. F. Codd at IBM in 1970.

The relational data model represents the database as a collection of simple two-dimensional tables called **tables** or **relations**. The rows of a relation are referred to as **tupples** and the columns are referred to as **attributes**. The relationship between the two relations is implemented through a common attribute in the relations and not by physical links or pointers.



| name | email | created_at | updated_at |
|---|---|---|---|
| Shawn | shawn@spbd.gov | 2012-12-10 T 10:00 UTC | 2013-02-16 T 14:00 UTC |
| Gus | gus@sbpd.gov | 2012-12-01 T 07:00 UTC | 2013-02-17 T 09:00 UTC |

***Fig 12: a relational database***

## VII. DATABASE NORMALIZATION

Normalisation is a process which we analyze and alter a database relation in order to get more concise and organized data structures. Normalised data is stable and has a natural structure. We call a relation normalized if:

- it does not contain any redundancy
- it does not cause maintenance problems
- it is an accurate representation of the data

Relations that aren't normalised contain non-atomic attributes and therefore can contain redundant information. Detailed planning of the ERM can help creating normalised relations. The following steps will explain how existing relations can be normalised step by step.

### VII.1 Dependencies

In order to be able to normalise a relation according to the three normal forms, we must first understand the concept of dependency between attributes within a relation.

#### VII.1.1 Functional dependency:

If A and B are attributes of relation R, B is functionally dependenton A (denoted A --> B), if each value of A in R is associated with exactly one value of B in R.

Example:

| **ID** | **Name** |
|---|---|
| S1 | Meier |
| S2 | Weber |

The attribute Name is functionally dependent of attribute ID (ID --> Name).

### VII.1.2 Identification key:

If every attribute B of R is functionally dependent of A, than attribute A is a primary key.

Example:

| ID | Name | Surname |
|---|---|---|
| S1 | Meier | Hans |
| S2 | Weber | Ueli |

Attribute ID is the identification key

### VII.1.3 Full functional dependency:

We talk about full functional dependency if attribute B is functional dependent on A, if A is a composite primary key and B is not already functional dependent on parts of A.

Example

| IDStudent | Name | IDProfessor | Grade |
|---|---|---|---|
| S1 | Meier | P2 | 5 |
| S2 | Weber | P1 | 6 |

The attribute Grade is fully functional dependent on the attributes IDStudent and IDProfessor.

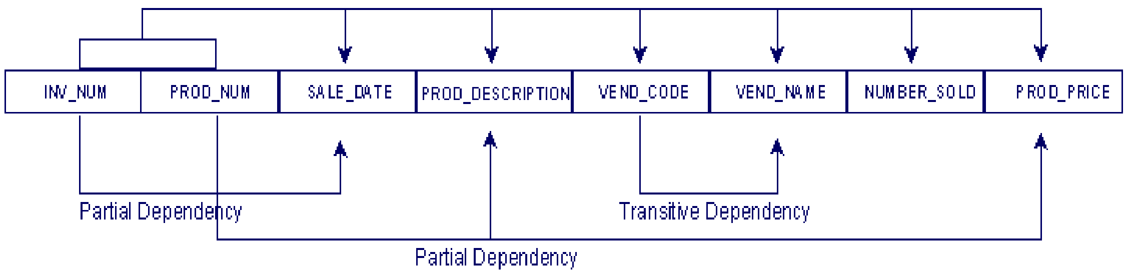
### VII.1.4 Transitive dependency:

If A determines B and B determines C then C is determined by (dependent on) A. We write A --> B and B --> C but not B --> A.

**Example:**

| ID | Name | Konto_Nr | Bank_Code_No | Bank |
|---|---|---|---|---|
| L1 | Meier | 1234-5 | 836 | UBS |
| L2 | Weber | 5432-1 | 835 | CS |

There is a transitive dependency between Bank_Code_No and Bank because Bank_Code_No is not the primary key of the relation.



## VII.2 First normal form (1NF)

**A table is in first normal form** (1NF) if a relation cannot have repeating fields or groups (no field must have more than one value): To do it, we have to:

a) Eliminate duplicative columns from the same table.

b) Create separate tables for each group of related data and identify each row with a unique column or set of columns (the primary key).

**Example**

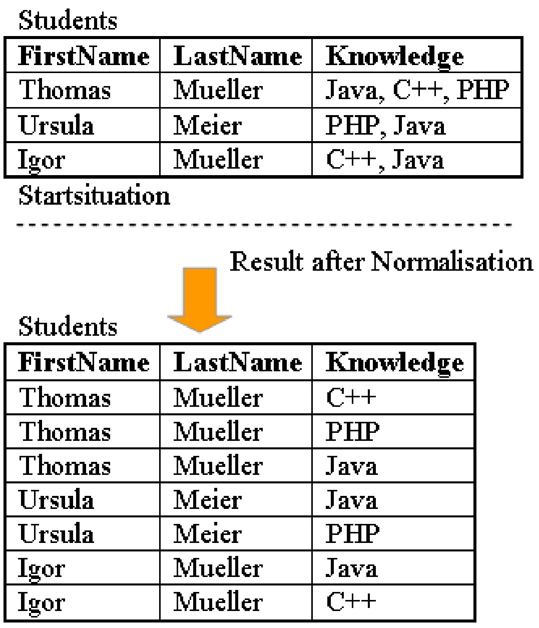
Student(Surname, Name, Skills)

The attribute Skills can contain multiple values and therefore the relation is not in the first normal form.

But the attributes Name and Surname are atomic attributes that can contain only one value.

Example First normal form

To get to the first normal form (1NF) we must create a separate tuple for each value of the multivalued attribute



Students

| FirstName | LastName | Knowledge |
|---|---|---|
| Thomas | Mueller | Java, C++, PHP |
| Ursula | Meier | PHP, Java |
| Igor | Mueller | C++, Java |

Startsituation

Result after Normalisation

Students

| FirstName | LastName | Knowledge |
|---|---|---|
| Thomas | Mueller | C++ |
| Thomas | Mueller | PHP |
| Thomas | Mueller | Java |
| Ursula | Meier | Java |
| Ursula | Meier | PHP |
| Igor | Mueller | Java |
| Igor | Mueller | C++ |

## VII.3 Second normal form (2NF)

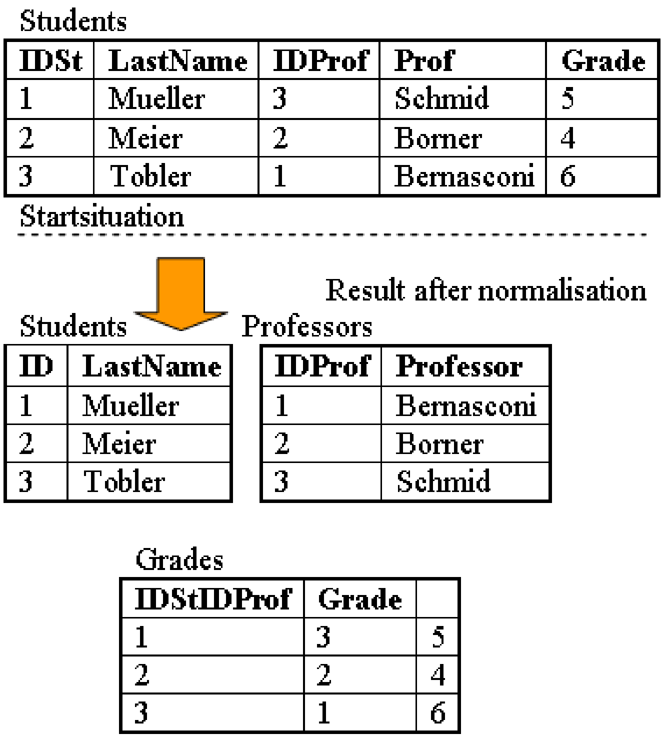
**A table is in second normal form** (2NF) if it is in 1NF and every non-key field must be functionally dependent on **all** of the key. To do it, one should:

a) Remove subsets of data that apply to multiple rows of a table and place them in separate tables.
b) Create relationships between these new tables and their predecessors through the use of foreign keys.

**Example**

A university uses the following relation:

***Student(IDSt, StudentName, IDProf, ProfessorName, Grade)***



Students

| IDSt | LastName | IDProf | Prof | Grade |
|---|---|---|---|---|
| 1 | Mueller | 3 | Schmid | 5 |
| 2 | Meier | 2 | Borner | 4 |
| 3 | Tobler | 1 | Bernasconi | 6 |

Startsituation

Result after normalisation

Students

| ID | LastName |
|---|---|
| 1 | Mueller |
| 2 | Meier |
| 3 | Tobler |

Professors

| IDProf | Professor |
|---|---|
| 1 | Bernasconi |
| 2 | Borner |
| 3 | Schmid |

Grades

| IDStIDProf | Grade | |
|---|---|---|
| 1 | 3 | 5 |
| 2 | 2 | 4 |
| 3 | 1 | 6 |

The attributes ***IDSt*** and ***IDProf*** are the identification keys. All attributes a single valued (1NF).

The following functional dependencies exist:

- ✓ The attribute ProfessorName is functionally dependent on attribute IDProf (IDProf --> ProfessorName)
- ✓ The attribute StudentName is functionally dependent on IDSt (IDSt --> StudentName)
- ✓ The attribute Grade is fully functional dependent on IDSt and IDProf (IDSt, IDProf --> Grade)

Example Second normal form

The table in this example is in first normal form (1NF) since all attributes are single valued. But it is not yet in 2NF. If student 1 leaves university and the tuple is deleted, then we loose all information about professor Schmid, since this attribute is fully functional dependent on the primary key IDSt. To solve this problem, we must create a new table Professor with the attribute Professor (the name) and the key IDProf. The third table Grade is necessary for combining the two relations Student and Professor and to manage the grades. Besides the grade it contains only the two IDs of the student and the professor. If now a student is deleted, we do not loose the information about the professor.

## VII.4 Third normal form (3NF)

**A table is in third normal form** (3NF) if it is in 2NF and There is no **transitive dependency**, that is an attribute depends on one or more other non-key attributes. We the have to remove columns that are not dependent upon the primary key.

**Example**

A bank uses the following relation:

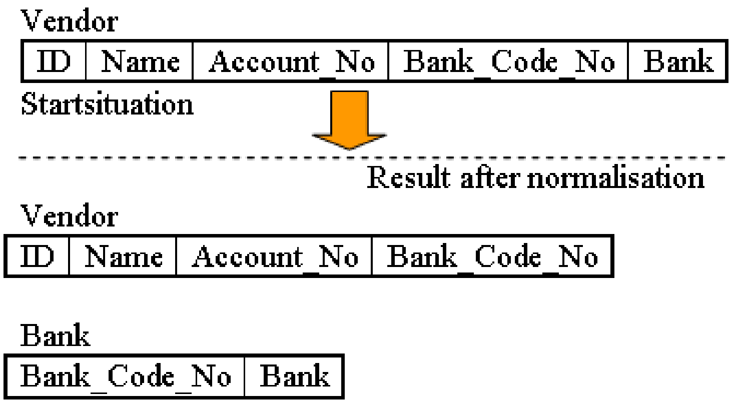
Vendor(ID, Name, Account_No, Bank_Code_No, Bank)

The attribute **ID** is the primary key. All attributes are single valued (1NF). The table is also in 2NF. The following dependencies exist:

1. Name, Account_No, Bank_Code_No are functionally dependent on ID (ID --> Name, Account_No, Bank_Code_No)
2. Bank is functionally dependent on Bank_Code_No (Bank_Code_No --> Bank)

Example Third normal form

The table in this example is in 1NF and in 2NF. But there is a transitive dependency between Bank_Code_No and Bank, because Bank_Code_No is not the primary key of this relation. To get to the third normal form (3NF), we have to put the bank name in a separate table together with the clearing number to identify it.



### VII.5 Fourth normal form (4NF)

A relation is in 4NF if it has no multi-valued dependencies. In practice we rarely need to apply the 4NF to a database.

## VIII. INTRODUCTION TO QUERIES

When referring to a database, a **query** is an option used to locate information within a database. For example you could query a database to find all related tables that match the state of "Utah" in the United States. The database could then return all people logged in the database that live in Utah. The most known and most used query language is SQL (Structured Query Language). A **form** is a page that contains a listing of fields that are required to be filled out.

A **report** is a file or view of data formatted in a way that allows the user to see a large amount of data from documents or a database. This helps the user get a quick overview of their data or allows them to present a lot of data easily.

### VIII.1 Introduction to SQL

SQL (Structured Query Language) is a database query language developed by IBM in the 1970s as a way of getting information into and out of relational DBMSs. A fundamental difference between SQL and standard programming languages is that SQL is declarative, that is, the user has to specify what kind of data are required from the database and the RDBMS is responsible for figuring out the way to retrieve it.

### VIII.2 Basic operations in a RDB

In a relational database, three basic operations are used to develop useful sets of data: ***selection, projection*** and ***join***.

- **The selection** operation retrieves certain records from a relation based on the user-specified criteria.
- **The projection** operation extracts fields from a relation, permitting the user to create new relations that contain only the required information.
- **The join** operation combines the data from the two relations based on a common column, providing the user with more information than is available in individual relations.

Together, these three operations are part of relational algebra. Relational database systems use a query language called Structured Query Language (SQL) to implement relational algebra operations.

### VIII.3 SQL Data Types

When the table is defined every field in it is assigned a data type. The type of a data value both defines and constrains the kinds of operations, which may be performed on it. Some of the most commonly used SQL data types are as follows:

- **CHAR(size):** It defines a fixed size-length character string (can contain letters, numbers and special characters), where size can be a maximum of 255.
- **VARCHAR(size)**: It defines a variable length character string (can contain letters, numbers and special characters) of up to size characters.
- **NUMBER(size):** It defines an integer-type data with maximum number of digits up to size specified in parenthesis.
- **DATE:** This type of data is used to store date. By default, the format is YYYY-MM-DD.
- **NUMBER(size, decimal):** It holds numbers with fractions. The maximum numbers of digits are specified in size. The maximum number of digits to the right of the decimal is specified in decimal.

## VIII.4 SQL Basic Commands

SQL commands can be divided into two main sublanguages—DDL and DML

- **Data Definition Language**: ***DDL*** is used to create and delete database and its objects. These commands are primarily used by the DBA during the building and removal phases of a database project. The most important DDL statements in SQL are as follows:
  - **CREATE TABLE**: To create a new table.
  - **ALTER TABLE**: To modify the structure of a table.
  - **DROP TABLE**: To delete a table.
- **Data Manipulation Language**: ***DML*** is used to retrieve, insert, modify and delete database information. These commands will be used by all database users during the routine operation of the database. The most important DML statements in SQL are the following:
  - **INSERT**: To insert data into a table.
  - **UPDATE**: To update data in a table.
  - **DELETE:** To delete data from a table.
  - **SELECT**: To retrieve data from a table.

***NOTE****: All SQL queries must be terminated by a semicolon (;) even if the statement extends over many lines.*

### VIII.4.1 CREATE TABLE Command

The CREATE TABLE command is used to define the structure of the table.

| **Syntax:** | **Example:** |
|---|---|
| CREATE TABLE <tablename> (<br><field1> <data type>,<br>< field2> <data type>, | CREATE TABLE EMPLOYEE(<br>Code NUMBER(5),<br>Deptt CHAR(10), |

| | |
|---|---|
| … … … … …<br>< fieldN> <data type>); | Name CHAR(20),<br>Address CHAR(100),<br>Telephone CHAR(8)<br>Salary NUMBER(8,2)); |

- The table and column names must start with a letter followed by letters, numbers or underscores.
- Avoid using SQL keywords as names for tables or columns (such as SELECT, CREATE, and INSERT).
- For each column, a name and a data type must be specified and the column name must be unique within the table definition.
- Each column definition should be separated with a comma.

### VIII.4.2 ALTER TABLE Command

The ALTER TABLE command allows a user to change the structure of an existing table.

- New columns can be added with the ADD clause.
- Existing columns can be modified with the MODIFY clause.
- Columns can be removed from a table by using the DROP clause.

**Syntax:**

```
ALTER TABLE <tablename>
<ADD | MODIFY | DROP column(s)>;
```

| | **Examples:** | **Explanation** |
|---|---|---|
| 1 | ALTER TABLE EMPLOYEE ADD Email CHAR(25); | command will add a new column, named Email, having a maximum width of 25<br>characters in the EMPLOYEE table. |
| 2 | ALTER TABLE EMPLOYEE MODIFY Name CHAR(25); | command will change the maximum width of the Name column to 25 characters in the EMPLOYEE table. |
| 3 | ALTER TABLE EMPLOYEE DROP Deptt; | command will delete the Deptt column from the EMPLOYEE table. |

### VIII.4.3 DROP TABLE Command

The DROP TABLE command removes the table definition (with all records).

- Columns can be removed from a table by using the DROP clause.

| **Syntax:** | **Examples:** |
|---|---|
| DROP TABLE <tablename>; | DROP TABLE EMPLOYEE; |

The above SQL command will delete the EMPLOYEE table.

### VIII.4.4 INSERT Command

The INSERT command is used to insert or add rows (records) into the specified table.

**Syntax:**

```
INSERT INTO <tablename> (
column1, column2, ..., columnN)
VALUES (value1, value2, ..., valueN);
```

| | **Examples** | **Explanation** |
|---|---|---|
| 1 | INSERT INTO EMPLOYEE (<br>Code, Deptt, Name, Address, Salary)<br>VALUES (101, 'RD01', 'Prince', 'Park Way', 15000); | example 1 will add a new record at the bottom of the EMPLOYEE table consisting of the values in parenthesis. |
| 2 | INSERT INTO EMPLOYEE<br>VALUES (102, 'RD01', 'Pankaj', 'Pitampura', 26062700, 8000); | |

Note that for each of the listed columns, a matching value must be specified. In case no column list is specified, then a value must be given for each column and in the same order as specified in the CREATE TABLE command.

### VIII.4.5 UPDATE Command

The UPDATE command is used for modifying attribute values of records in a table.

**Syntax:**

```
UPDATE <tablename>
SET column1 = value1
[, column2 = value2]
... ... ... ... ...
[, columnN = valueN]
[WHERE <condition>];
```

Note that components specified inside the square brackets [] are optional.

| | **Examples** | **Explanation** |
|---|---|---|
| 1 | UPDATE EMPLOYEE<br>SET Salary = Salary + 1000; | example 1 command will update (in our case, increments) the Salary field with 1000 for all the records. |
| 2 | UPDATE EMPLOYEE<br>SET Salary = Salary + 1000<br>WHERE Deptt = 'RD01'; | Example 2 command will increment the Salary column with 1000 for only those rows that comply with condition specified in WHERE clause (Deptt = 'RD01'). |

### VIII.4.6 DELETE Command

The DELETE command is used to delete all or selected records from the specified table.

| **Syntax:** | **Example:** |
|---|---|
| DELETE FROM <tablename><br>[WHERE <condition>]; | DELETE FROM EMPLOYEE<br>WHERE Salary > 8000; |

***NOTE:*** *If WHERE condition is not used in the DELETE command, then all the records from the specified table will be deleted.*

### VIII.4.7 SELECT command

The SELECT statement is used to query the database and retrieve selected data.

**Syntax:**

```
SELECT <column1, column2, column3,...., columnN>
FROM <tablename>
[WHERE <condition>]
[GROUP BY <column1, column2, column3,...., columnN>]
[HAVING <condition>]
[ORDER BY <column1, column2, column3,...., columnN [ASC|DESC]>];
```

To select all the columns of a table, use * instead of column list with SELECT.

| | **Examples** | **Explanation** |
|---|---|---|
| **1** | SELECT Code, Name, Salary<br>FROM EMPLOYEE; | The SELECT statement selects the values of the three specified columns from the EMPLOYEE table. This operation is called projection. |
| **2** | SELECT *<br>FROM EMPLOYEE<br>WHERE Salary > 7500; | The SELECT statement selects all those columns from EMPLOYEE table in which the Salary column contains a value greater than 7500. This operation is called selection. |
| | SELECT *<br>FROM EMPLOYEE<br>WHERE Salary > 7500<br>ORDER BY Name DESC; | the SELECT statement displays the result in a descending order by the attribute Name. |

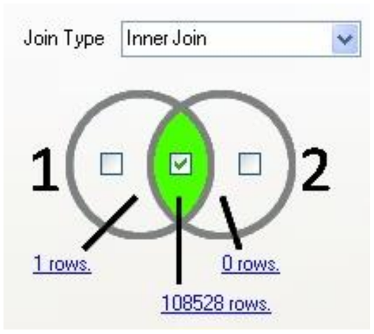
The **ORDER BY** clause specifies a sorting order in which the result tuples of a query are to be displayed; ***DESC*** specifies a descending order. By default, **ORDER BY** arranges the result set in ascending order (whether one uses ***ASC*** or not).

## VIII.5 SQL Joins

An SQL JOIN clause is used to combine rows from two or more tables, based on a common field between them. SQL specifies two main types of JOIN: Inner Join and Outer Join

### VIII.5.1 Inner Join

The INNER JOIN creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate. The query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row.



**Syntax:**

The basic syntax of **INNER JOIN** is as follows:

```
SELECT table1.column1, table2.column2...
FROM table1
INNER JOIN table2
```

```
ON table1.common_filed = table2.common_field;
```

**Example:**

Consider the following two tables,

(a) CUSTOMERS table is as follows:

| ID | NAME | AGE | ADDRESS | SALARY |
|---|---|---|---|---|
| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |
| 2 | Khilan | 25 | Delhi | 1500.00 |
| 3 | kaushik | 23 | Kota | 2000.00 |
| 4 | Chaitali | 25 | Mumbai | 6500.00 |
| 5 | Hardik | 27 | Bhopal | 8500.00 |
| 6 | Komal | 22 | MP | 4500.00 |
| 7 | Muffy | 24 | Indore | 10000.00 |

(b) Another table is ORDERS as follows:

| OID | DATE | ID | AMOUNT |
|---|---|---|---|
| 102 | 2009-10-08 | 3 | 3000 |
| 100 | 2009-10-08 | 3 | 1500 |
| 101 | 2009-11-20 | 2 | 1560 |
| 103 | 2008-05-20 | 4 | 2060 |

Now, let us join these two tables using INNER JOIN as follows:

```
SQL> SELECT ID, NAME, AMOUNT, DATE
   FROM CUSTOMERS
   INNER JOIN ORDERS
   ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

This would produce the following result:

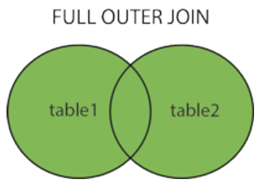
| ID | NAME | AMOUNT | DATE |
|---|---|---|---|
| 3 | kaushik | 3000 | 2009-10-08 |
| 3 | kaushik | 1500 | 2009-10-08 |
| 2 | Khilan | 1560 | 2009-11-20 |
| 4 | Chaitali | 2060 | 2008-05-20 |

### VIII.5.2 Outer join

Inner joins eliminate the rows that do not match with a row from the other table. Outer joins, however, return all rows from at least one of the tables or views mentioned in the FROM clause, as long as those rows meet any WHERE or HAVING search conditions. All rows are retrieved from the left table referenced with a ***left outer join***, and all rows from the right table referenced in a ***right outer join***. All rows from both tables are returned in a full outer join.

SQL Server uses the following ISO keywords for outer joins specified in a FROM clause:

- ✓ LEFT OUTER JOIN or LEFT JOIN
- ✓ RIGHT OUTER JOIN or RIGHT JOIN
- ✓ FULL OUTER JOIN or FULL JOIN



The following example JOINs the **region** and **branch** tables on the **region_nbr** column. Here are the contents of the tables:

**Table: REGION**

| region_nbr | region_name |
|---|---|
| 100 | East Region |
| 200 | Central Region |
| 300 | Virtual Region |
| 400 | West Region |

**Table: BRANCH**

| branch_nbr | branch_name | region_nbr | employee_count |
|---|---|---|---|
| 108 | New York | 100 | 10 |
| 110 | Boston | 100 | 6 |
| 212 | Chicago | 200 | 5 |
| 404 | San Diego | 400 | 6 |
| 415 | San Jose | 400 | 3 |

This SQL Statement with OUTER JOIN is executed:

```
 SELECT region.region_nbr, region.region_name, branch.branch_nbr, branch.branch_name
FROM dbo.region
LEFT OUTER JOIN dbo.branch
ON branch.region_nbr = region.region_nbr
ORDER BY region.region_nbr
```

Here is the result. Note that the "Virtual Region" is included in the results even though it has no rows in the **branch** table. This is the difference between the INNER JOIN and OUTER JOIN.

| region_nbr | region_name | branch_nbr | branch_name |
|---|---|---|---|
| 100 | East Region | 108 | New York |
| 100 | East Region | 110 | Boston |
| 200 | Central Region | 212 | Chicago |
| 300 | Virtual Region | NULL | NULL |
| 400 | West Region | 404 | San Diego |
| 400 | West Region | 415 | San Jose |

**Exercise normalization**

The following table is already in first normal form (1NF). Convert this table to the third normal form (3NF) using the techniques you learned in this topic.

A table with the students and their grades in different topics.

| UnitID | StudentID | Date | TutorID | Topic | Room | Grade | Book | TutEmail |
|---|---|---|---|---|---|---|---|---|
| U1 | St1 | 23.02.03 | Tut1 | GMT | 629 | 4.7 | Deumlich | tut1@fhbb.ch |
| U2 | St1 | 18.11.02 | Tut3 | GIn | 631 | 5.1 | Zehnder | tut3@fhbb.ch |
| U1 | St4 | 23.02.03 | Tut1 | GMT | 629 | 4.3 | Deumlich | tut1@fhbb.ch |
| U5 | St2 | 05.05.03 | Tut3 | PhF | 632 | 4.9 | Dümmlers | tut3@fhbb.ch |
| U4 | St2 | 04.07.03 | Tut5 | AVQ | 621 | 5.0 | SwissTopo | tut5@fhbb.ch |