

#### 4. 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.

This is the basic structure of the database that organizes the data, defines how the data are stored, accessed and the relationships between the data. Types of Data Models:

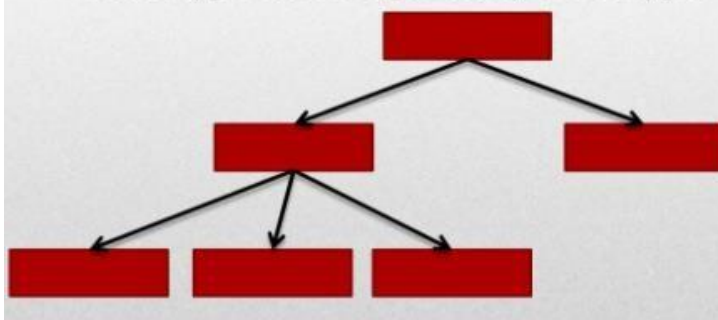
##### Types of Data models:

- Hierarchical Model
- Network Model
- Entity-Relationship(E-R) Model
- Relational Model
- Object-Oriented Model
- Object-Relational Model

##### 1. Hierarchical Data Model:

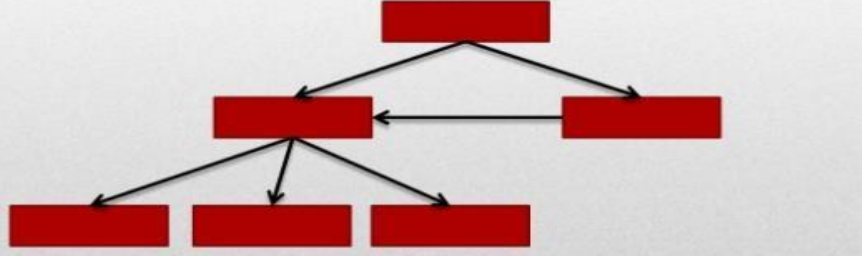
www.binils.com

- Data is represented as a tree.
- A record type can belong to only one owner type but a owner type can belong to many record type.



##### 2. Network Model:

- It is modified version of Hierarchical Data Model where it allows more general connections among the nodes as well.
- They are difficult to use but are more flexible than hierarchical databases.



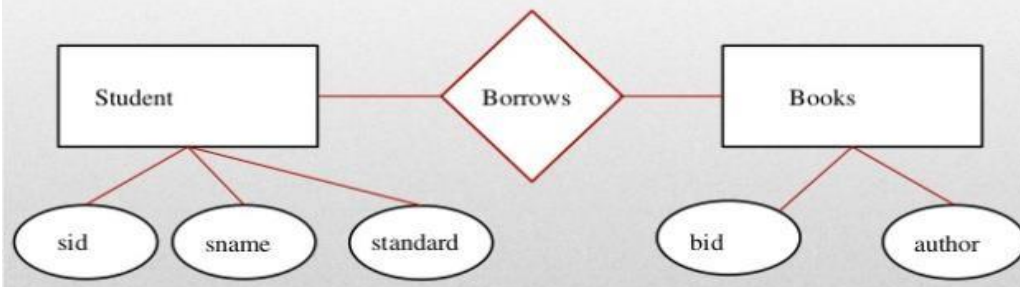
### 3. Relational Model:

- It is a lower level model that uses a collection of tables to represent both data and relationships among those data.
- Each table has multiple columns, depending on the number of attributes, and each column has a unique name.

Student		
sid	sname	Standard
A-101	Ramesh	11
A-102	Kriti	10
A-103	Laxmi	12

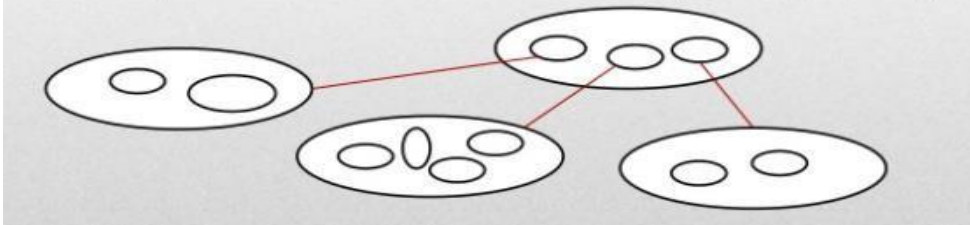
### 4. Entity relationship model

- It is a high level model based on the need of the organization.
- Its entities are distinguishable from other objects and relationship is an association among several entities.



### 5. Object Oriented Model

- It represents entity sets as class and a class represents both attributes and the behavior of the entity.
- Instance of a class is an object.
- The internal part of the object is not externally visible.
- One object communicates with the other by sending messages.

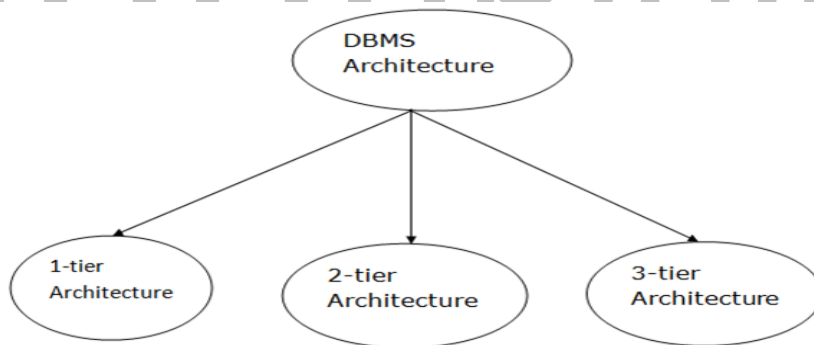


## 6. Object Relational model:

It combines the feature of relational model and object oriented model.

## 5. DBMS Architecture

- DBMS architecture depends upon how users are connected to the database to get their request done.

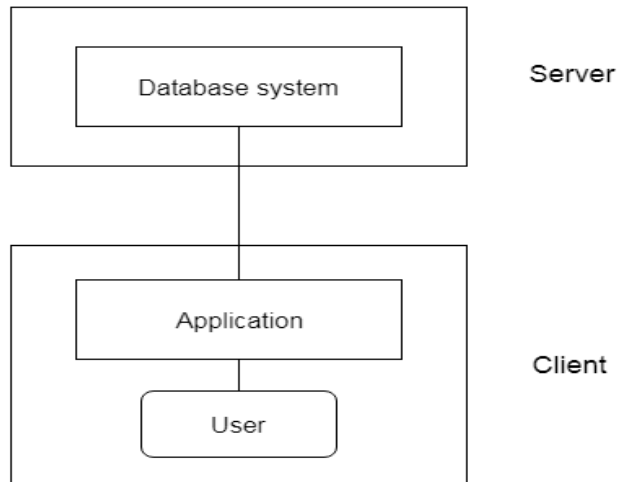


Database architecture can be seen as a single tier or multi-tier. But logically, database architecture is of two types like: **2-tier architecture** and **3-tier architecture**.

### 1-Tier Architecture

- In this architecture, the database is directly available to the user. It means the user can directly sit on the DBMS and uses it.
- Any changes done here will directly be done on the database itself. It doesn't provide a handy tool for end users.
- The 1-Tier architecture is used for development of the local application, where programmers can directly communicate with the database for the quick response.

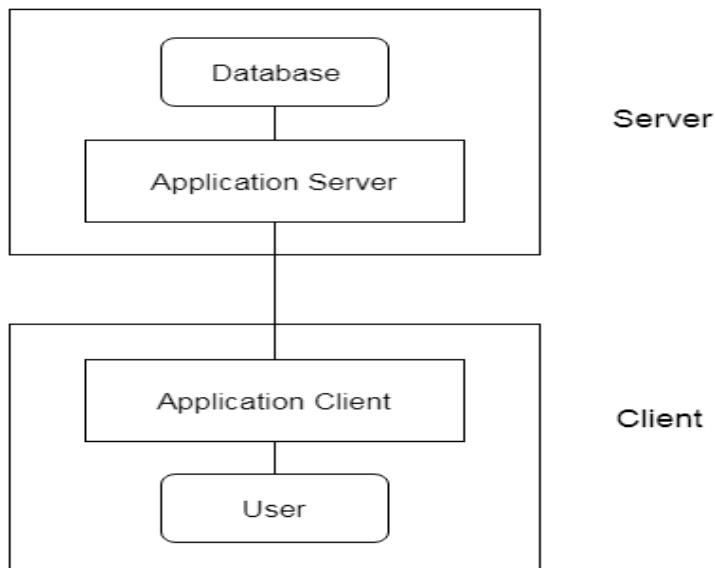
## 2-Tier Architecture



- The 2-Tier architecture is same as basic client-server. In the two-tier architecture, applications on the client end can directly communicate with the database at the server side. For this interaction, API's like: **ODBC**, **JDBC** are used.
- The user interfaces and application programs are run on the client-side.
- The server side is responsible to provide the functionalities like: query processing and transaction management.
- To communicate with the DBMS, client-side application establishes a connection with the server side.

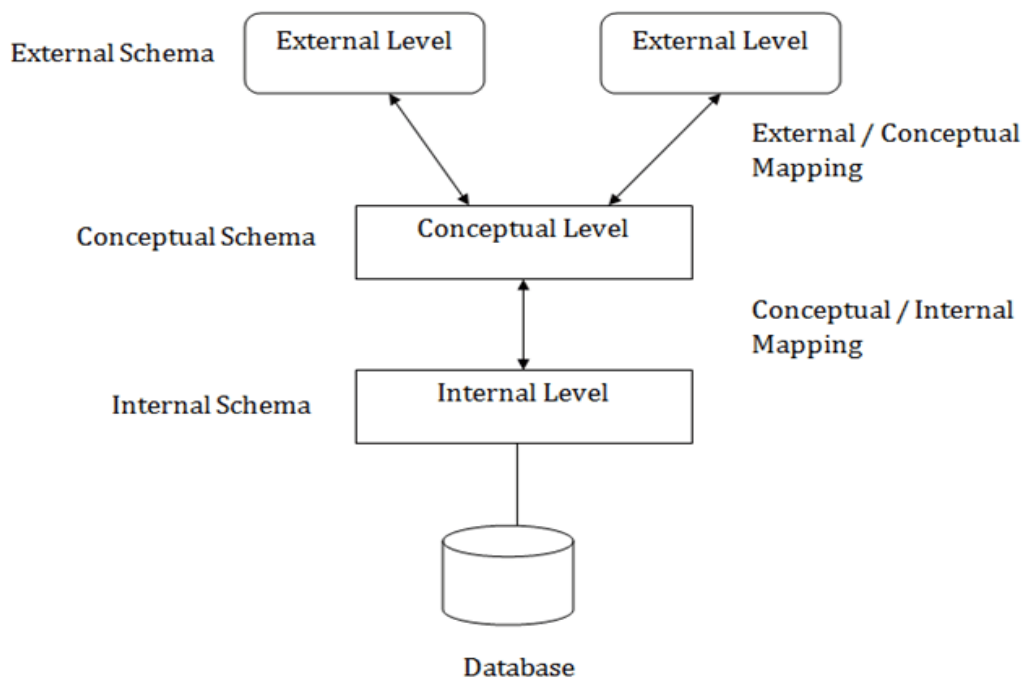
## 3-Tier Architecture

- The 3-Tier architecture contains another layer between the client and server. In this architecture, client can't directly communicate with the server.
- The application on the client-end interacts with an application server which further communicates with the database system.
- End user has no idea about the existence of the database beyond the application server. The database also has no idea about any other user beyond the application.
- The 3-Tier architecture is used in case of large web application.



### Three Schema Architecture

- The three schema architecture is also called ANSI/SPARC architecture or three-level architecture.
- This framework is used to describe the structure of a specific database system.
- The three schema architecture is also used to separate the user applications and physical database. The three schema architecture contains three-levels. It breaks the database down into three different categories.



**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 views. 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.

## 5.1 COMPONENTS OF DBMS

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

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

**1. Application programmers:** Professionals who write application programs and using these application programs they interact with the database system

**2. Sophisticated users :** These user interact with the database system without writing programs, But they submit queries to retrieve the information

**3. Specialized users:** Who write specialized database applications to interact with the database system.

**4. 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

**5. 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:

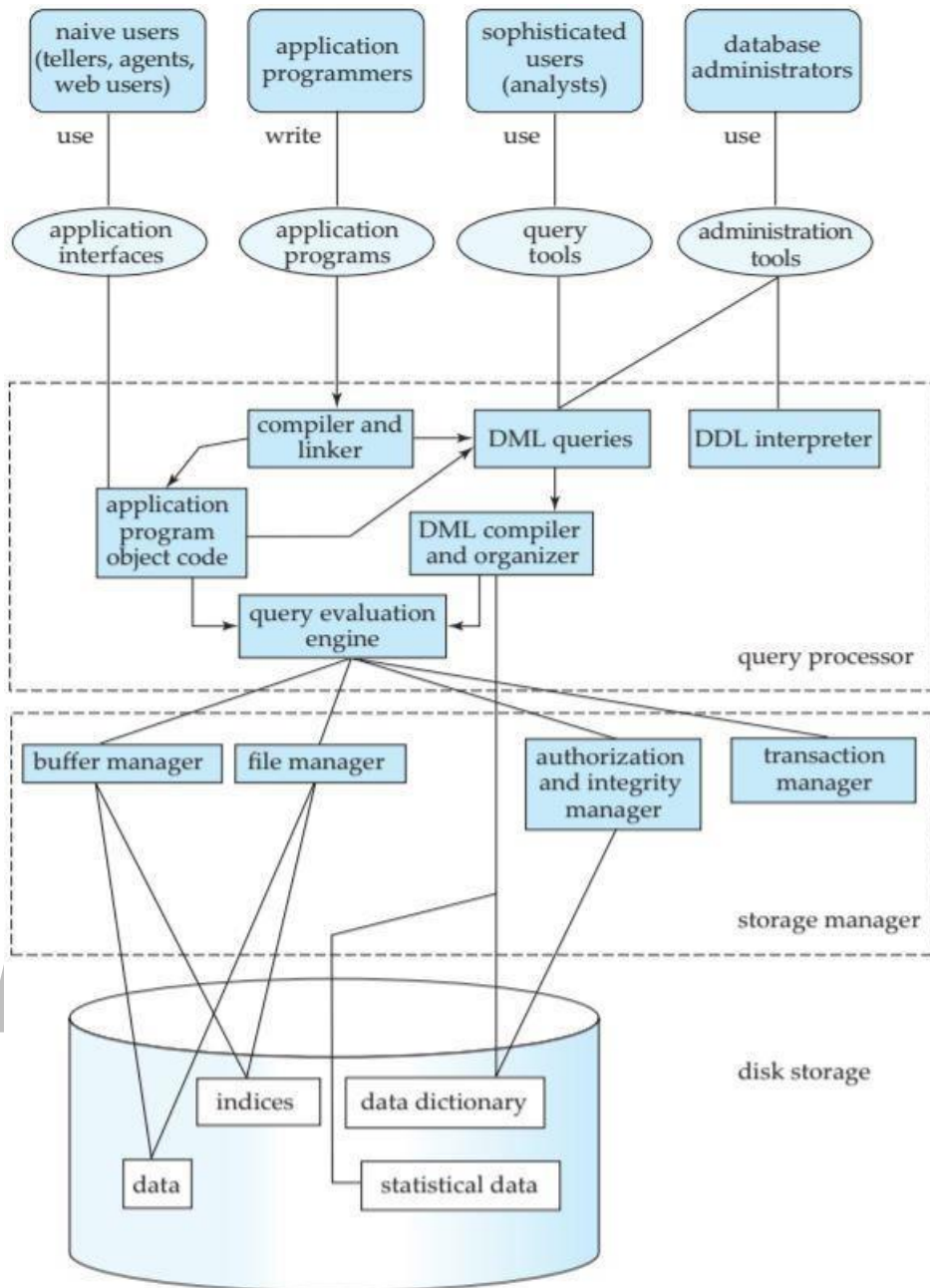
- Interaction with the file manager
- Efficient storing, retrieving and updating of data

#### **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





## 10. 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.
- This is the simplest approach to embed SQL statements directly into the source code files that will be used to create an application. This technique is referred to as embedded SQL programming.
- 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.
- High level programming language compilers cannot interpret SQL statements.
- Hence source code files containing embedded SQL statements must be preprocessed before compiling.
- Thus each SQL statement coded in a high level programming language source code file must be prefixed with the keywords EXEC SQL and terminated with either a semicolon or the keyword END\_EXEC.

### Connection to Database:

- 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.

### Host variables

- Database manager cannot work directly with high level programming language variables.
- Instead, it must be special variables known as host variables to move data between an application and a database.

#### Two types of host variables.

- Input host variables: Transfer data to database.
- Output host variable :Receives data from database

Host variables are ordinary programming language., They must be defined within a special section known as declare section.

```
EXEC SQL BEGIN DECLARE SECTION;
```

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

- Each host variable must be assigned a unique name even though declared in different declaration section.

The following code is a simple embedded SQL program, written in C.

- 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.

```
main() {  
    EXEC SQL BEGIN DECLARE SECTION;  
    int OrderID, CustID;  
    char SalesPerson[10], Status[6];  
    EXEC SQL END DECLARE SECTION;  
  
    printf ("Enter order number: ");  
    scanf ("%d", &OrderID);  
  
    EXEC SQL SELECT CustID, SalesPerson, Status FROM  
Orders WHERE OrderID = :OrderID INTO :CustID,  
:SalesPerson, :Status;  
  
    printf ("Customer number: %d \n", CustID);  
    printf ("Salesperson: %s \n", SalesPerson);  
    printf ("Status: %s \n", Status);  
}
```

## 12. DYNAMIC SQL

- **Static or Embedded SQL** are SQL statements in an application that do not change at runtime and, therefore, can be hard-coded into the application. Dynamic SQL is SQL statements that are constructed at runtime; for example, the application may allow users to enter their own queries.
- **Dynamic SQL** is a programming technique that enables you to build SQL statements dynamically at runtime. You can create more general purpose, flexible applications by using dynamic SQL

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.

### 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;
```

### Example

```
#include <stdio.h>
#include <conio.h>
int main(){
EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
int STD_ID;
char *STD_NAME;
int CLASS_ID;
char *sql_stmt;
char *sql_query;
EXEC SQL END DECLARE SECTION;
EXEC WHENEVER NOT FOUND GOTO error_msg1;
EXEC WHENEVER SQLERROR GOTO error_msg2;
```

```
printf("Enter the Student name:");
scanf("%s", STD_Name);
printf("Enter the Class ID:");
scanf("%d", &CLASS_ID);
sql_stmt = "SELECT STD_ID FROM STUDENT ";
if (strcmp(STD_NAME, ' ') != 0)
{
sql_stmt = sql_stmt || " WHERE STD_NAME = :STD_NAME";
}
else if (CLASS_ID > 0)
{
sql_stmt = sql_stmt || " WHERE CLASS_ID = :CLASS_ID";
}
EXEC SQL PREPARE sql_query FROM :sql_stmt;
EXEC SQL EXECUTE sql_query;
printf("STUDENT ID:%d", STD_ID);
exit(0);
```

Output

**Assume the table**

**STUDENT**

STD_ID	STD_NAME	CLASS_ID	CITY	CONTACT_NO
CS001	ARUN	101	NAGERCOIL	XXX
CS002	ASHA	102	VALLIYOR	XXX
CS025	MAHESH	202	TIRUNELVELI	XX

Enter the Student name : ASHA

**(STD\_NAME = ASHA)**

Enter the Class ID: 202

**(CLASS\_ID=202)**

(If give one valid string as student name the query will be constructed like this:)

```
SELECT STD_ID FROM STUDENT WHERE STD_NAME = :STD_NAME
```

**And the output will be**

**CS002**

**Else**

Enter the Student name: PRIYA

**(STD\_NAME = PRIYA)**

If CLASS\_ID > 0 the query will be framed like

```
SELECT STD_ID FROM STUDENT WHERE CLASS_ID = :CLASS_ID
```

**And the output will be**

**CS025**

[www.binils.com](http://www.binils.com)

## 1. INTRODUCTION TO 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.

A DBMS is a collection of software programs that allows a user to define datatypes, structures, constraints, store data permanently, modify and delete operations.

DBMS is basically a software used to add, modify, delete, select data from database.

In simpler words, DBMS is a collection of interrelated data and software programs to access those data.

### 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

## 2.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.

**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

## 3. 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, this is called data abstraction. So developer hides complexity from user and store abstract view of data.

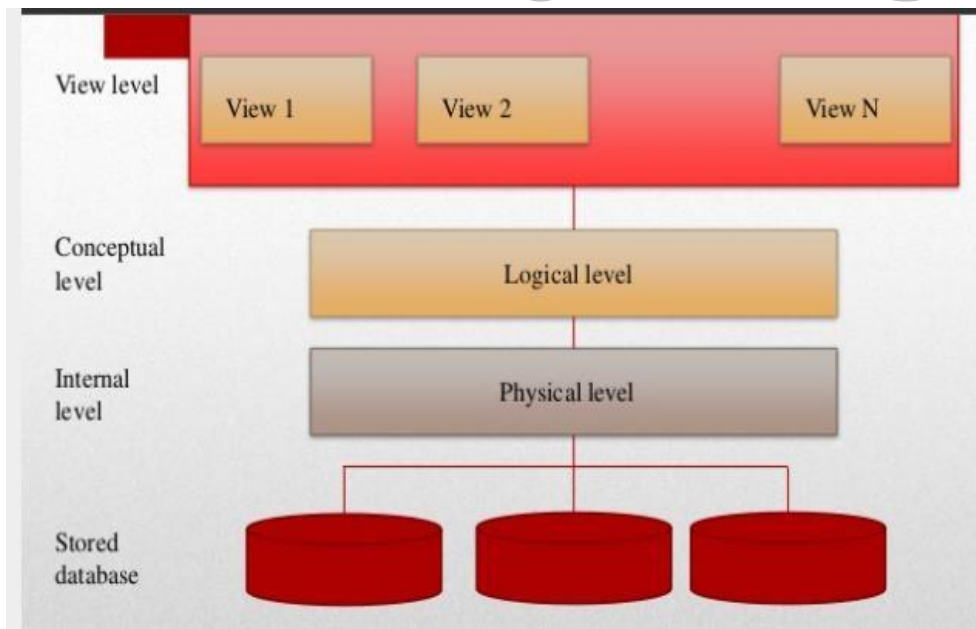
Data abstraction has three level of abstractions

- Physical 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 unviewuser see data and there may be many views of one data like chart and graph.



## 7. KEYS AND THEIR USE

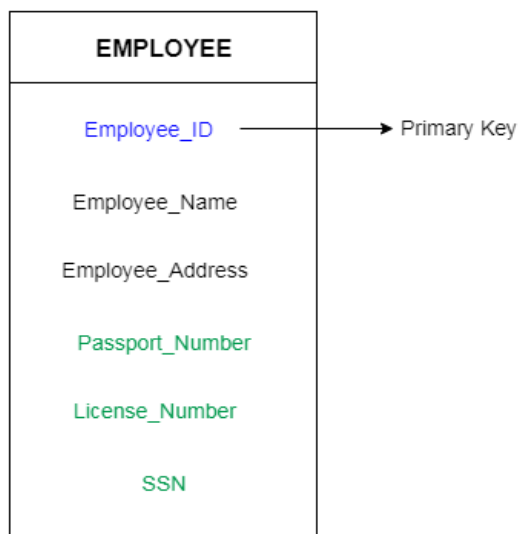
**Key:** An attribute or set of attributes whose values uniquely identify each entity in an entity set is called a key for that entity set.

**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).

### 1. Primary key

- It is the first key which is used to identify one and only one instance of an entity uniquely. An entity can contain multiple keys as we saw in PERSON table. The key which is most suitable from those lists becomes a primary key.
- In the EMPLOYEE table, ID can be primary key since it is unique for each employee. In the EMPLOYEE table, we can even select License\_Number and Passport\_Number as primary key since they are also unique.
- For each entity, selection of the primary key is based on requirement and developers.

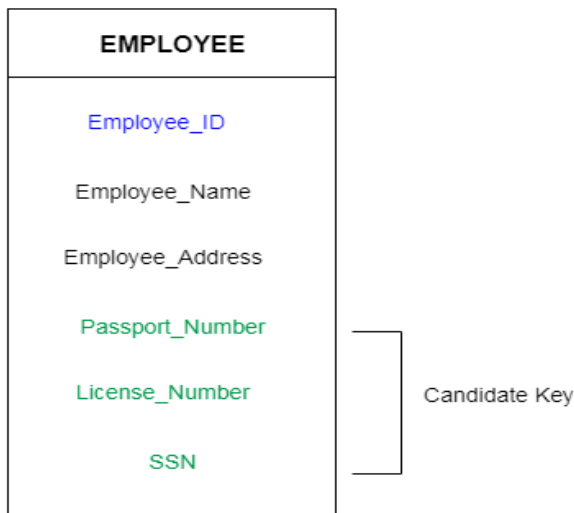


**2. Candidate Key:** There may be two or more attributes or combinations of attributes that uniquely identify an instance of an entity set. These attributes or combinations of attributes are called candidate keys.

## 2. Candidate key

- A candidate key is an attribute or set of an attribute which can uniquely identify a tuple.
- The remaining attributes except for primary key are considered as a candidate key. The candidate keys are as strong as the primary key.

**For example:** In the EMPLOYEE table, id is best suited for the primary key. Rest of the attributes like SSN, Passport\_Number, and License\_Number, etc. are considered as a candidate key.



**3. 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.

### 3. Super Key

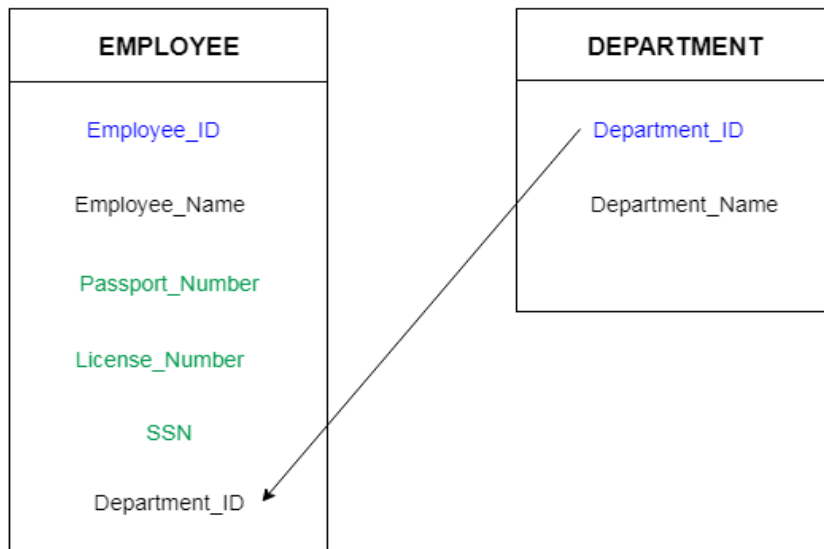
Super key is a set of an attribute which can uniquely identify a tuple. Super key is a superset of a candidate key.

- **For example:** In the above EMPLOYEE table, for (EMPLOYEE\_ID, EMPLOYEE\_NAME) the name of two employees can be the same, but their EMPLOYEE\_ID can't be the same. Hence, this combination can also be a key.
- The super key would be EMPLOYEE-ID, (EMPLOYEE\_ID, EMPLOYEE-NAME), etc.

**4. 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.

#### 4. Foreign key

- Foreign keys are the column of the table which is used to point to the primary key of another table.
- In a company, every employee works in a specific department, and employee and department are two different entities. So we can't store the information of the department in the employee table. That's why we link these two tables through the primary key of one table.
- We add the primary key of the DEPARTMENT table, Department\_Id as a new attribute in the EMPLOYEE table.
- Now in the EMPLOYEE table, Department\_Id is the foreign key, and both the tables are related.



5. **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**.

#### 7.1 RELATIONAL INTEGRITY CONSTRAINTS

- Relational Integrity constraints in DBMS are referred to conditions which must be present for a valid relation. These Relational constraints in DBMS are derived from the rules in the mini-world that the database represents.
- There are many types of Integrity Constraints in DBMS. Constraints on the Relational database management system is mostly divided into three main categories are:

- Domain Constraints
- Key Constraints
- Referential Integrity Constraints

### Domain Constraints

Attributes have specific values in real-world **scenario**. For example, age can only be a positive integer

- Domain constraints specify that within each tuple, and the value of each attribute must be unique. This is specified as data types which include standard data types integers, real numbers, characters, Booleans, variable length strings, etc.

### Key Constraints

- An attribute that can uniquely identify a tuple in a relation is called the key of the table. The value of the attribute for different tuples in the relation has to be unique.

#### Example:

In the given table, CustomerID is a key attribute of Customer Table. It is most likely to have a single key for one customer, CustomerID =1 is only for the CustomerName =" Google".

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

### Referential Integrity Constraints

- 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 constraints in DBMS are based on the concept of Foreign Keys.

- A foreign key is an important attribute of a relation which should be referred to in other relationships. 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.

**Example:**

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

Customer

InvoiceNo	CustomerID	Amount
1	1	\$100
2	1	\$200
3	2	\$150

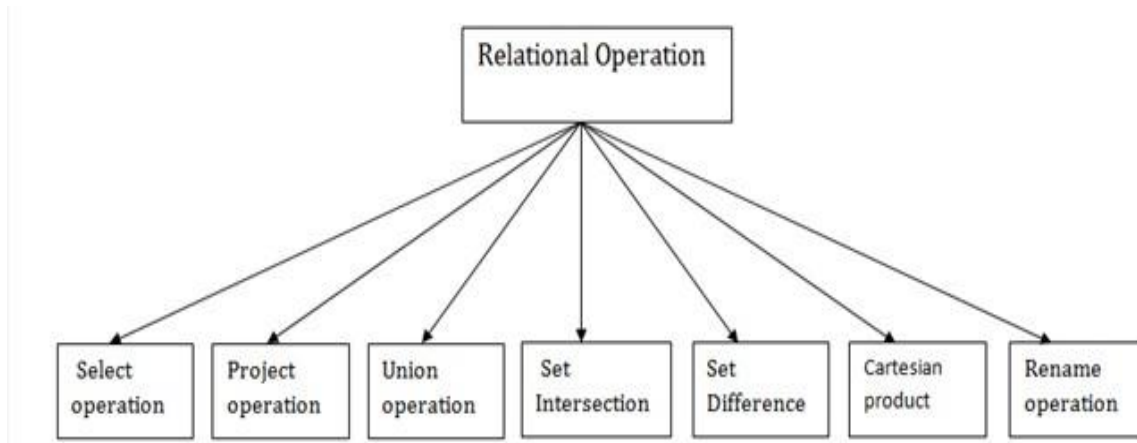
Billing

In the above example, we have 2 relations, Customer and Billing.

Tuple for CustomerID =1 is referenced twice in the relation Billing. So we know CustomerName=Google has billing amount \$300

## 8. RELATIONAL ALGEBRA

Relational algebra is a procedural query language. It gives a step by step process to obtain the result of the query. It uses operators to perform queries. which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries.



### 1. Select Operation:

- The select operation selects tuples that satisfy a given predicate.
- It is denoted by sigma ( $\sigma$ ).
- Notation:  $\sigma_p(r)$

**Where:**  $\sigma$  is used for selection prediction

$r$  is used for relation

$p$  is used as a propositional logic formula which may use connectors like: AND OR and NOT.

These relational can use as relational operators like =,  $\neq$ ,  $\geq$ ,  $<$ ,  $>$ ,  $\leq$ .

Table Name : LOAN

BRANCH_NAME	LOAN_NO	AMOUNT
Downtown	L-17	1000
Redwood	L-23	2000
Perryride	L-15	1500
Downtown	L-14	1500
Mianus	L-13	500
Roundhill	L-11	900
Perryride	L-16	1300

**Input:**

$\sigma$  BRANCH\_NAME="perryride" (LOAN)

**Output:**

BRANCH_NAME	LOAN_NO	AMOUNT
Perryride	L-15	1500
Perryride	L-16	1300

## 2. Project Operation:

- This operation shows the list of those attributes that we wish to appear in the result. Rest of the attributes are eliminated from the table.
- It is denoted by  $\pi$ .
- Notation:  $\pi$  A1, A2, An (r)
- **Where A1, A2, A3** is used as an attribute name of relation r.



**Example: CUSTOMER RELATION**

NAME	STREET	CITY
Jones	Main	Harrison
Smith	North	Rye
Hays	Main	Harrison
Curry	North	Rye
Johnson	Alma	Brooklyn
Brooks	Senator	Brooklyn

**Input:**

TI NAME, CITY (CUSTOMER)

**Output:**

NAME	CITY
Jones	Harrison
Smith	Rye
Hays	Harrison
Curry	Rye
Johnson	Brooklyn
Brooks	Brooklyn

**3. Union Operation:**

- Suppose there are two tuples R and S. The union operation contains all the tuples that are either in R or S or both in R & S.
- It eliminates the duplicate tuples. It is denoted by  $\cup$ .
- Notation:  $R \cup S$

**DEPOSITOR RELATION**

<b>CUSTOMER_NAME</b>	<b>ACCOUNT_NO</b>
Johnson	A-101
Smith	A-121
Mayes	A-321
Turner	A-176
Johnson	A-273
Jones	A-472
Lindsay	A-284

**BORROW RELATION**

<b>CUSTOMER_NAME</b>	<b>LOAN_NO</b>
Jones	L-17
Smith	L-23
Hayes	L-15
Jackson	L-14
Curry	L-93
Smith	L-11
Williams	L-17

**Input:**

```
Π CUSTOMER_NAME (BORROW) ∪ Π CUSTOMER_NAME (DEPOSITOR)
```

**Output:**

CUSTOMER_NAME
Johnson
Smith
Hayes
Turner
Jones
Lindsay
Jackson
Curry
Williams
Mayes

#### 4. Set Intersection:

- Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in both R & S.
- It is denoted by intersection  $\cap$ .

Notation:  $R \cap S$

**Example:** Using the above DEPOSITOR table and BORROW table

**Input:**

$\Pi$  CUSTOMER\_NAME (BORROW)  $\cap$   $\Pi$  CUSTOMER\_NAME (DEPOSITOR)

**Output:**

**CUSTOMER\_NAME**

Smith

Jones

## 5. 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  $s$ .

Notation:  $R - S$

**Example:** Using the above DEPOSITOR table and BORROW table

**Input:**

$\Pi$  CUSTOMER\_NAME (BORROW) -  $\Pi$  CUSTOMER\_NAME (DEPOSITOR)

**Output:**

**CUSTOMER\_NAME**

Jackson

Hayes

Willians

Curry


## 6. Cartesian product

- The Cartesian product is used to combine each row in one table with each row in the other table. It is also known as a cross product.
- It is denoted by X.

Notation: E X D

Example:

EMPLOYEE



EMP_ID	EMP_NAME	EMP_DEPT
1	Smith	A
2	Harry	C
3	John	B

DEPARTMENT

DEPT_NO	DEPT_NAME
A	Marketing
B	Sales
C	Legal

Input:

EMPLOYEE X DEPARTMENT

Output:

EMP_ID	EMP_NAME	EMP_DEPT	DEPT_NO	DEPT_NAME
1	Smith	A	A	Marketing
1	Smith	A	B	Sales
1	Smith	A	C	Legal
2	Harry	C	A	Marketing
2	Harry	C	B	Sales
2	Harry	C	C	Legal
3	John	B	A	Marketing
3	John	B	B	Sales
3	John	B	C	Legal

2	Harry	C	C	Legal
3	John	B	A	Marketing
3	John	B	B	Sales
3	John	B	C	Legal

## 7. Rename Operation:

The rename operation is used to rename the output relation. It is denoted by  $\rho$  ( $\rho$ ).

**Example:** We can use the rename operator to rename STUDENT relation to STUDENT1.

```
 $\rho(\text{STUDENT1}, \text{STUDENT})$ 
```

## 7. Joins operation in relational algebra

Join operation in relational algebra is a combination of a Cartesian product followed by which satisfy certain condition. A Join operation combines two tuples from two different relations, if and only if a given condition is satisfied.

There are different types of join operations.

**(I) Natural Join ( $\bowtie$ )** A result of natural join is the set of tuples of all combinations in R and S that are equal on their common attribute names.

It is denoted by  $\bowtie$ .

### Natural Join ( $\bowtie$ )

Natural join is a binary operator. Natural join between two or more relations will result set of all combination of tuples where they have equal common attribute.

Let us see below example

Emp			Dep	
(Name	Id	Dept_name)	(Dept_name	Manager)
A	120	IT	Sale	Y
B	125	HR	Prod	Z
C	110	Sale	IT	X
D	111	IT		

Emp  $\bowtie$  Dep

Name Id Dept\_name Manager

A	120	IT	X
C	110	Sale	Y
D	111	IT	X

Consider the following example to understand natural Joins.

### EMPLOYEE

EMP_ID	EMP_NAME
1	Ram
2	Varun
3	Lakshmi

### SALARY

EMP_ID	SALARY
1	50000
2	30000
3	25000

□ EMP\_NAME, SALARY (EMPLOYEE ⋈ SALARY)

**Output:**

EMP_NAME	SALARY
Ram	50000
Varun	30000
Lakshmi	25000



## (II) Outer Join

Outer joins are used to include all the tuples from the relations included in join operation in the resulting relation.

An outer join is of three types:

1. Left outer join
2. Right outer join
3. Full outer join

Consider the example **EMPLOYEE**

EMP_NAME	STREET	CITY
Ram	Civil line	Mumbai
Varun	S.G.Road	Kolkata
Lakshmi	C.G.Road	Delhi
Hari	AnandNagar	Hyderabad

EMP_NAME	BRANCH	SALARY
Ram	Infosys	10000
Varun	Wipro	20000
Neha	HCL	30000
Hari	TCS	50000

**(i) Left outer join (⋈)**

In Left outer join, all the tuples from the Left relation, say R, are included in the resulting relation. If there are some tuples in relation R which are not matched with tuple in the Right Relation S, then the attributes of relation R of the resulting relation become NULL.

**EMPLOYEE ⋈ FACT\_WORKERS**

Output:

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Varun	S.G.Road	Kolkata	Wipro	20000
Hari	AnandNagar	Hyderabad	TCS	50000
Lakshmi	C.G.Road	Delhi	NULL	NULL

**(ii) Right outer join (⋈)**

In Right outer join, all the tuples from the Right relation, say S, are included in the resulting relation. If there are some tuples in relation S which are not matched with tuple in the Right Relation R, then the attributes of relation S of the resulting relation become NULL.

**EMPLOYEE ⋈ FACT\_WORKERS**

Output :

EMP_NAME	BRANCH	SALARY	STREET	CITY
Ram	Infosys	10000	Civil line	Mumbai

Varun	Wipro	20000	S.G.Road	Kolkata
Hari	TCS	50000	AnandNagar	Hyderabad
Neha	HCL	30000	NULL	NULL

### (iii) Full outer join

Full outer join is the combination of both left outer join and right outer join. It contains all the tuples from both relations.

For example

EMPLOYEE  $\bowtie$  FACT\_WORKERS

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Varun	S.G.Road	Kolkata	Wipro	20000
Hari	AnandNagar	Hyderabad	TCS	50000
Lakshmi	C.G.Road	Delhi	NULL	NULL
Neha	NULL	NULL	HCL	30000

### (iv) Theta ( $\theta$ ) Join

Theta join is denoted by the symbol  $\theta$ . It combines those tuples from different relations which satisfies the condition.

**Notation –  $R1 \bowtie_{\theta} R2$**

Where R1 and R2 are relations with n numbers of attributes such that the attributes do not have anything in common, it means  $R1 \cap R2 = \Phi$ .

<b>Student</b>		
<b>Stud_ID</b>	<b>Name</b>	<b>Std</b>
1	Ram	10
2	Shyam	11

<b>Subjects</b>	
<b>Class</b>	<b>Subject</b>
10	Math
10	English
11	Music
11	Sports

<b>SID</b>	<b>Name</b>	<b>Std</b>	<b>Class</b>	<b>Subject</b>
1	Ram	10	10	Math
1	Ram	10	10	English
2	Shyam	11	11	Music
2	Shyam	11	11	Sports

## RELATIONAL ALGEBRA EXAMPLE SET 2

### Player relation

Player Id	Team Id	Country	Age	Runs	Wickets
1001	101	India	25	10000	300
1004	101	India	28	20000	200
1006	101	India	22	15000	150
1005	101	India	21	12000	400
1008	101	India	22	15000	150
1009	103	England	24	6000	90
1010	104	Australia	35	1300	0
1011	104	Australia	29	3530	10
1012	105	Pakistan	28	1421	166
1014	105	Pakistan	21	3599	205

### Deposit relation

Acc. No.	Cust-name
A 231	Rahul
A 432	Omkar
R 321	Sachin
S 231	Raj
T 239	Sumit

Loan No.	Cust-name
P-3261	Sachin
Q-6934	Raj
S-4321	Ramesh
T-6281	Anil

### 1. SELECT

1. Find all tuples from player relation for which country is India.

$\sigma_{\text{country} = \text{India}}(\text{Player})$

2. Select all the tuples for which runs are greater than or equal to 15000.

$\sigma_{\text{runs} \geq 15000}(\text{Player})$

3. Select all the players whose runs are greater than or equal to 6000 and age is less than 25

$\sigma_{\text{runs} \geq 6000 \wedge \text{age} < 25}(\text{Player})$

### 2. PROJECT

1. List all the countries in Player relation.

$\pi_{\text{country}}(\text{Player})$

2. List all the team ids and countries in Player Relation

$\pi_{\text{Team Id, Country}}(\text{Player})$

### 3. Set Difference Operation (-)

1. Find all the customers having an account but not the loan.

$\pi_{\text{cust-name}}(\text{Depositor}) - \pi_{\text{cust-name}}(\text{Borrower})$

Result –

Cust-name
Rahul
Omkar
Sumit

2. Find all the customers having a loan but not the account.

$\pi_{\text{cust-name}}(\text{Borrower}) - \pi_{\text{cust-name}}(\text{Depositor})$

Result

Cust-name
Ramesh
Anil

#### 4. Cartesian Product Operation (X)

**Example**

Employee-Schema = { Emp-id, Name }

Emp-Id	Name
101	Sachin
103	Rahul
104	Omkar
106	Sumit
107	Ashish

Project-Schema = { Proj-name }

Proj-name
DBMS 1
DBMS 2

Find R = Employee X Project

Solution –

R-Schema = {Emp-id, Name, Proj-name}

Emp-Id	Name	Proj-name
101	Sachin	DBMS 1
101	Sachin	DBMS 2
103	Rahul	DBMS 1
103	Rahul	DBMS 2
104	Omkar	DBMS 1
104	Omkar	DBMS 2
106	Sumit	DBMS 1
106	Sumit	DBMS 2
107	Amit	DBMS 1
107	Amit	DBMS 2

Solution  
 $\rho(\text{CustomerList}, \text{Customer})$ .



### Set Intersection Operation ( $\cap$ )

Relation A

Id	Name
101	Rahul
104	Anil

Relation B

Id	Name
101	Anil
104	Anil

**1. Find all tuples that are common in relations A and B.**

Solution – We have to find

$$P = A \cap B$$

So, P is

Id	Name
104	Anil

**2. Find names of customers having an account and loan (using Borrower and Deposit, given at start of this post).**

Solution – We have to apply set intersection operation in Borrower and Deposit Relation (given at start of this post).

These can be represented as

$$P = \text{Borrower} \cap \text{Deposit}.$$

So, P is

Cust-name
Sachin
Raj

## 6. 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.

### 6..1 RELATIONAL MODEL

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.

The data in an RDBMS is stored in database objects which are called as **tables**. This table is basically a collection of related data entries and it consists of numerous columns and rows. The following program is an example of a CUSTOMERS table –

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

Every table is broken up into smaller entities called fields. The fields in the CUSTOMERS table consist of ID, NAME, AGE, ADDRESS and SALARY.

- A **field is a column** in a table that is designed to maintain specific information about every record in the table.
- A **record** is also called as a row of data is each individual entry that exists in a table. For example, there are 7 records in the above CUSTOMERS table. Following is a single row of data or record in the CUSTOMERS table –

1	Ramesh	32	Ahmedabad	2000.00
---	--------	----	-----------	---------

A record is a horizontal entity in a table.

- A **NULL value** in a table is a value in a field that appears to be blank, which means a field with a NULL value is a field with no value.

## 6.2 OPERATIONS IN RELATIONAL MODEL

Four basic update operations performed on relational database model are

- Insert, update, delete and select.
- **Insert** is used to insert data into the relation
- **Delete** is used to delete tuples from the table.
- **Modify** allows you to change the values of some attributes in existing tuples.
- **Select** allows you to choose a specific range of data.

Whenever one of these operations are applied, integrity constraints specified on the relational database schema must never be violated

### Insert Operation

The insert operation gives values of the attribute for a new tuple which should be inserted into a relation.

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive
4	Alibaba	Active



## Update Operation

You can see that in the below-given relation table CustomerName= 'Apple' is updated from Inactive to Active.

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive
4	Alibaba	Active



CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Active
4	Alibaba	Active

## Delete Operation

To specify deletion, a condition on the attributes of the relation selects the tuple to be deleted.

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Active
4	Alibaba	Active




CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
4	Alibaba	Active

In the above-given example, CustomerName= "Apple" is deleted from the table.

## Select Operation

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
4	Alibaba	Active



CustomerID	CustomerName	Status
2	Amazon	Active

In the above-given example, CustomerName="Amazon" is selected

## 9. SQL FUNDAMENTALS

### What is SQL?

- SQL stands for **Structured Query Language**
- 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 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. 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).

### 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":

#### 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":

<b>LastName</b>	<b>FirstName</b>	<b>Address</b>	<b>City</b>
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 Language types:

Structured Query Language (SQL) as we all know is the database language by the use of which we can perform certain operations on the existing database and also we can use this language to create a database. SQL uses certain commands like Create, Drop, Insert, etc. to carry out the required tasks.

These SQL commands are mainly categorized into four categories as:

1. DDL – Data Definition Language
2. DQL – Data Query Language
3. DML – Data Manipulation Language
4. DCL – Data Control Language

Though many resources claim there to be another category of SQL clauses **TCL – Transaction Control Language**. So we will see in detail about TCL as well.

1. **DDL (Data Definition Language)**: DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database.

### Examples of DDL commands:

- **CREATE** – is used to create the database or its objects (like table, index, function, views, store procedure and triggers).

- **DROP** – is used to delete objects from the database.
- **ALTER**-is used to alter the structure of the database.
- **TRUNCATE**–is used to remove all records from a table, including all spaces allocated for the records are removed.
- **COMMENT** –is used to add comments to the data dictionary.
- **RENAME** –is used to rename an object existing in the database.

### **DQL (Data Query Language) :**

1. **DQL** statements are used for performing queries on the data within schema objects. The purpose of the DQL Command is to get some schema relation based on the query passed to it.

#### **Example of DQL:**

- **SELECT** – is used to retrieve data from the database.

2. **DML(Data Manipulation Language):** The SQL commands that deals with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements.

#### **Examples of DML:**

- **INSERT** – is used to insert data into a table.
- **UPDATE** – is used to update existing data within a table.
- **DELETE** – is used to delete records from a database table.

3. **DCL(Data Control Language):** DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions and other controls of the database system.

#### **Examples of DCL commands:**

- **GRANT**-gives user's access privileges to the database.
- **REVOKE**-withdraw user's access privileges given by using the GRANT command.

4. **TCL(transaction Control Language):** TCL commands deal with the transaction within the database.

#### **Examples of TCL commands:**



- **COMMIT**– commits a Transaction.
- **ROLLBACK**– rolls back a transaction in case of any error occurs.
- **SAVEPOINT**– sets a savepoint within a transaction.
- **SET TRANSACTION**– specify characteristics for the transaction.

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


### SQL-DML Query examples

#### *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.

www.binils.com

#### *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 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
```

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

```
SELECT DISTINCT Company FROM Orders
```

**Result:**

Company
Sega
W3Schools
Trio

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

**WHERE clause:**

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'
```

### LIKE Condition

#### 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%'
```

## INSERT INTO

### *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	

## UPDATE

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

### Syntax

www.binils.com

```
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:**

<b>LastName</b>	<b>FirstName</b>	<b>Address</b>	<b>City</b>
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Stien 12	Stavanger

## **DELETE**

### *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:**

<b>LastName</b>	<b>FirstName</b>	<b>Address</b>	<b>City</b>
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**

<b>LastName</b>	<b>FirstName</b>	<b>Address</b>	<b>City</b>
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
```

## **ORDER BY**

### 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 in MS Access and SQL server

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

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



<b>COUNT(*)</b>	Returns the number of selected rows
<b>FIRST(column)</b>	Returns the value of the first record in a specified field
<b>LAST(column)</b>	Returns the value of the last record in a specified field
<b>MAX(column)</b>	Returns the highest value of a column
<b>MIN(column)</b>	Returns the lowest value of a column
<b>STDEV(column)</b>	This function is used to compute statistical standard deviation from sample data.
<b>STDEVP(column)</b>	This function is used to compute statistical standard deviation for an entire population data.

www.binils.com

### *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**

<b>Function</b>	<b>Description</b>
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

#### **GROUP BY**

## 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:

[www.binils.com](http://www.binils.com)

```
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...

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

### SQL example :

**Consider** the employee database , where the primary keys are Underlined.

Employee(empname,street,city)

Works(empname,companyname,salary)

Company(companyname,city)

Manages(empname,management)

Write the SQL query for the following requests

1) Find the names of all employees who work for Company A

**Ans:**

Select empname from Works Where companyname='Company A'

2) Find the names, street addresses and cities of residence of all employees who work for Company A and earn more than 200000 per annum.

**Ans:**

select \* from Employee where empname in

(select empname from Works where companyname='company A' and salary>200000)

**3)Find the names of all employees in this database who live in the same city as the company for which they work.**

**Ans:**

select E.empname from Employee as E, Works as W, Company as C where

E.empname=W.empname and E.city=C.city and

W.company\_name=C.company\_name

**4)Find the names of all employees who earn more than every employees of Company**

**B**

**Ans:**

select empname from Works where salary > all

(select salary from Works where companyname='Company B')

**5. Delete all tuples in the works relation for employees of Company B**

**Ans**

Delete from Works where companyname='Company B'