`Instance & Schema`:

`Instance` : The collection of information stored in a database at a particular moment is called **Instance** of the database.

`Schema` : The overall design of the database is called the **Schema** of the database.

- `Schema` :
- `Physical Schema` : @ the Physical level.
- `Logical Schema` : @ the Logical level.

- `Subschema` : @ the View level, a DB can have multiple schemas called **subschemas**, that describe different views of the database.

- `Physical data independence` : An **application program** is said to exhibit physical data independence *if it doesn't depend on Physical Schema* and thus need not be re-written if the physical schema changes.

- Off all the above different types of schemas, **Logical Schema** is the most important, in terms of its impact on the applications program as the programmer construct applications using these schemas only.

- **Physical Schema** is hidden beneath the Logical Schema and can easily be changed without affecting the applications program.

`Data Models`:

`Data Model` : It's a **collection of conceptual tools** used to describe the data, data relationships, data semantics, and consistency constraints.

- A data model provides a way to describe the design of database at the physical, logical and view levels.

`Data manipulation Language (DLM)`:

`DLM` : It's a language that enable users to access or manipulate data as organised by the appropriate Data Model.

- `DLMs` :
- `Procedural DLMs` : user must specify **What** data & **How** to get those data.

- `Declarative (or Nonprocedural) DLMs` : user need specify ONLY **What** data is needed. `Ex.: SQL query language`

- `Query` : A statement requesting the retrieval of information.

- `Query Language` : A portion of DLM that involves information retrieval.

- The `query processor` component of the Database system translates DLM queries into sequences of actions at the Physical level of the Database System.

`Data-Definition language (DDL) :`

`DDL` : We specify a **database schema** by a set of definitions expressed by a special language called **Data-definition Language (DDL)**.

- The `DDL` is also used to specify additional properties of the data.
- `Data Storage & Definition Language` : A set of statements in a special type of **DDL** used to specify the **storage structure and access methods** used by the database system.
  These statements define the implementation details of the database schemas (which are hidden from the users).
- The **DDL** provides the facility to specify **consistency constraints** and the database system these constraints every time the database is updated.
- The o/p of DDL is placed in **Data Dictionary** which contains *metadata*.
- `Data Dictionary` is considered to be a special type of **TABLE** that can *only* be accessed by the database system itself (not a regular user).
  *The database consults the **Data Dictionary** before reading or modifying actual data.*
- `atomicity`

- `consistency`
- `durability`

`RELATIONAL DATABASES:`

- A `Relational Model` hides low-level implementation from database developers and users.
- `Application Programs:` SQL **does not support** actions such as input from users, output to displays, or communication over network.
  Such computation and actions must be written in *host language* (like **C, C++, Java**) with embedded SQL queries that access the data in the DB.
  Programs that are used to interact w/ the DB in this fashion are called **Application Programs**.
- `DML precompiler` : It converts the DML statements to normal procedure calls in *host language* (C, C++, Java).
- Database design mainly involves the design of Database Schemas.
- `Relationship` : An association among several entities.
- `Entity set` : Set of all entities of the same type.
- `Relationship set` : Set of all relationships of the same type.
- `UML` : Unified Modelling Language is one of the several ways to represent Entity-Relationship (ER) Diagram.
  In UML, `entity sets` are represented by rectangular boxes with the entity-name on it and its attributes below.
  `Relationship sets` are represented by Diamond box connecting a pair of related entity sets.
- `Mapping cardinalities` : The no. of entities to which another entity can be associated with via a relationship set.
- `Normalization` : An approach of designing schemas that stores information w/o unnecessary

redundancy but still allows us retrieve information easily. The schemas are then termed as **Normal forms**.

- `Functional Dependencies` : An approach to determine whether a relation schema is of desirable normal form.

- `Storage manager` : Its a component of the database system that provides the interface b/w the low-level data stored in the database and the application program and queries submitted to the system.

- The **Storage manager** converts the DML commands into low-level file-system commands. The Storage Manager is responsible for storing, retrieving and updating data in the database.

- **Storage Manager** consists of:

    - `Authorization & Integrity Manager`
    - `Transaction Manager`
    - `File Manager`
    - `Buffer Manager`

- **Query Processor** :

    - `DDL interpretor` : Interprets DDL statements and record definitions in Data Dictionary.
    - `DML compiler` : Converts DML statements into low-level instructions that the query evaluation engine understands.
    - `Query Evaluation Engine` : It evaluates low-level instructions generated by the DML compiler.

- **Transaction** : A collection of operations that perform a single logical functions in a database application.

- **Failure Recovery**: Failure detection and restoration of DB to a prior state before failure.

- **Concurrency control manager**: It controls the interaction among the concurrent transactions to ensure consistency.

- **Transaction manager** controls the Concurrency Control Manager and Recovery manager.

- **Information Retrieval**: Unlike the rigidly structured data in Relational Databases, textual data is unstructured & querying of **unstructured textual data** is called ***information retrieval***.

- **Object-oriented Data Models**: An extension of E-R model with notions of *encapsulation, methods/functions & object-identity*

- **Object-Relational Data Model**: A data model that combines the concepts of *object-oriented data model* and *relational data model*.

- **Semi-structured Data Model**: A model which permits the specification of data where **individual data item of same type** may have different attributes.

- In Relational Model,

    - **relation** is used to refer to a Table
    - **tuple** refers to a Row in table
    - **attribute** refer to a Column of table

- **CREATE**:

```sql
create table r(
A1 D1,
A2 D2,
A3 D3,
.....,
A_n D_n,
<integrity_constraint_1>,
......,
<integrity_constraint_k>);

-- for example
create table department(
dept_name varchar(10),
building varchar(10),
budget numeric(12,2),
primary key(dept_name)); -- the integrity constraint
```

- **INSERT**:

```sql
insert into instructor
values(2017, 'Potter', 'Rustom',19862);
```

- **DELETE**: used to delete *tuples* from a *relation*

```sql
-- this will delete all tuples from student relation
delete from student;
```

- **DROP**: to remove a complete *relation* from the SQL Database. It deletes all tuples in the relation and also its schema. After the *relation* is dropped, no tuples can be inserted into the relation unless it is *re-created* using `create table` command.

```sql
-- deletes all tuples in r and its schema as well
drop table r;
```

- **ALTER TABLE**: It is used to add *attributes* to an existing relation. All tuples in the relation are assigned **NULL** values for the newly added attribute.

We can drop an *attribute* from a relation using the **DROP** command as well (but many DBMS don’t allow individual drop of an attribute, rather they will allow dropping the whole relation using `drop table r` command)

```sql
-- add an attribute A of datatype D into the relation r
alter table r add A D;

-- drops the attribute A from all tuples in relation r
alter table r drop A;
```

- **DISTINCT**: It is used to eliminate duplicate tuples from a relation.

ALL keyword is used to specify explicitly that the duplicates are not removed (also, it is the default feature of SQL unless specified otherwise)

```
-- removes duplicates from the query result & returns only the distinct dept_name.
-- The result of this query contains the each value of dept_name attribute at most once.

select distinct dept_name from instructors;
```

- **SELECT**: The *select* clause may also contain arithmetic expressions involving `+,-,*,/`. But, if we used an arithmetic expression in a select clause then the resultant attribute does not have the a name.

```
-- here, the third attribute of the o/p relation won't have a name
select ID, name, dept_name, salary*1.1 from instructor;
```

- The **WHERE** clause allows us to select only those tuples in the result *relation* of the **FROM** clause that satisfies a given predicate.

`and, or, not` logical connectives are allowed in the **WHERE** clause.

```
select name from instructor
where dept_name = 'Comp_Sc' and salary >= 70000;
```

- **LIKE**: SQL expresses patterns using `like` comparision operator
    - `'Intro%'` : matches any string beginning with `Intro`
    - `'%Intro%'` : Matches any string containing the substring `Intro`
    - `___` : matches any string of exactly 3 characters
    - `___%` : matches any string of atleast 3 characters

```
-- it returns the dept_name whose building name contains
-- the substring 'Watson'

select dept_name from department
where building like '%Watson%';
```

- **ORDER BY**: Causes the tuples in the result relation of a query to appear in **sorted order**. By default, `order by` clause lists elements in **ascending order**.; so use `desc` or `asc` for descending and ascending order respectively.

```
select name from instructor
where dept_name = 'Physics'
order by name;

----------------------


select instructor.*
from instructor, teaches
where instructor.ID = teaches.ID
order by instructor.ID;
```

```
---------------------
-- gives the entire 'instructor' relation in descending order
-- of 'salary' and if the 'salary' of two tuples matches then
-- they are ordered in ascending order of 'name'

select * from instructor
order by salary desc, name asc;
```

- **BETWEEN / NOT BETWEEN**: Used to simplify **where** clause that specify an expression to be in a range of values.

```
select name from instructor
where salary between 90000 and 100000;
-- same as below code

select name from instructor
where salary >= 90000 and salary <= 100000;

-------------------------------------

select name from instructor
where salary not between 90000 and 100000;
-- same as below code

select name from from instructor
where salary <= 90000 or salary >= 100000;
```

- **UNION / UNION ALL**: The **union** operation automatically eliminates the duplicates unlike the **select** clause.

If we want to retain all duplicates then use **union all**

```

-- UNION: Duplicates are removed automatically

(
select course__id from section
where (semester, year) = ('Fall', 2016)
)
union
(
select course_id from section
where (semester, year) = ('Spring', 2017)
);


-- UNION ALL: Duplicates are NOT eliminated

(
select course_id from section
where (semester, year) = ('Fall', 2016)
)
union all
(
select course_id from section
where (Semester, year) = ('Spring', 2017)
);
```

- **INTERSECT / INTERSECT ALL**:
- **EXCEPT / EXCEPT ALL**: Outputs all operations in the first relation that does not appear in second relation i.e. it performs the **set difference**.

The **except** operation automatically eliminates duplicates in inputs **before** performing set difference, so to retain the duplicates we must use **except all**

```
-- Finds all courses taught in Fall'16 but not in Spring'17 sem.

(select course_id
from section
where (semester, year) = ('Fall', 2016))
except
(select course_id
from section
where (semester, year) = ('Spring', 2017));
```

- **Set Comparision**:
  - `= some` is identical to `in`
  - `<> all` is identical to `not in`
  - `<> some` is NOT identical to `not in`
  - `= all` is NOT identical to `in`

```
-- Give the name of instructor whose salary is greater than
-- ATLEAST one instructor in Biology dept.

select distinct T.name
from instructor as T, instructor as S
where T.salary > S.salary and S.dept_name = 'Biology';

-------------

select name
from instructor
where salary > some (select salary
from instructor
where dept_name = 'Biology');

-- Both queries above give the same o/p relation
```

MODIFICATION OF THE DATABASE

- **DELETE** :
  - We can delete only whole tuples, we can't delete values on only particular attributes.
  - `delete from r where P` : the delete statement finds all tuples `t` in relation `r` where

`P(t)` is `true` and then deletes those tuples.

  - `delete from r` : deletes all tuples in relation `r` (but keeps an empty relation r).

```sql
-- deletes all tuples from instructor relation but still keeps the relation
delete from instructor;

-- deletes all tuples where dept_name is 'Finance'
delete from instructor
where dept_name = 'Finance';

-- deletes all tuples where salary is between 90K to 100K
delete from instructor
where salary between 90000 and 100000;

-- finds the dept_name located in Watson building
-- & deletes all tuples in instructor with those dept_name
delete from instructor
where dept_name in (select dept_name
from department
where building = 'Watson');

--
```