DISCLAIMER

These are my personal notes which I prepared during my placement season. They are not professional and are not 100% accurate. So please use them as reference and always cross-verify things. Also if you liked my notes or if they have helped you in any way then please consider subscribing to my YouTube channel (https://youtube.com/@withloveshivangi) and follow me on Instagram (@withloveshivangi).

Thank you and good luck for your journey.

Owned by: Shivangi Singh

Date: 22 July,2023

Camlin Page
Date / /

DBMS core:

→ **DBMS**: a system that allows users to create & maintain a db. perfoms operation like insert, delete & updating.
- overcomes problem of data inconsistency (same data having diff values at diff place), data redundancy (repeated data)
- convenient, organised & secure.

→ **RDBMS: (Relational DBMS)**
- stores data in form of tables.
- more efficient than DBMS, as DBMS stores data as files. Eg: MySQL, oracle.

→ issues with traditional file-based system:
- time consuming to scan all, tedious, slow,
- data redundancy, inconsistent.
- unorganised.

→ Languages in DBMS:

• DDL (data definition lang): defines database.
- create, alter, drop, truncate, rename.
- alter: adds a new column to DB.
- drop: removes entire table, irreversible
- truncate: deletes all tuples (row), fastest and cant be rollback.

• DML (Data Manipulation Lang): manipulates the data present in db.
- select, update, insert, delete.

- Data control language (DCL): deals w/ user permissions & controls.
  - Grant - grants access
  - Revoked - takes back access.

- data transaction language: commands that deals with transaction of database.
  - commit: used to permanently save the changes done in transaction in database.
  - ROLLBACK: undo the transactions that have not been saved in the db.

- ACID Properties:

1. Atomicity: either execute whole query or execute nothing at all. reflect any update to entire dB.

2. consistency: data should be consistent/ correct after transactions

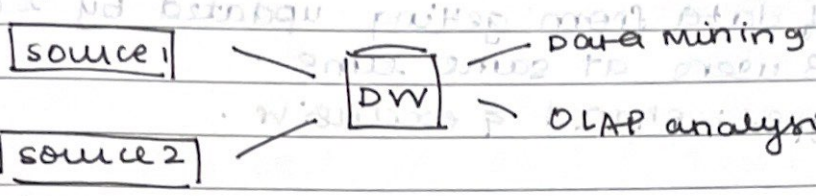
3. Isolation: multiple transactions can occur concurrently w/o leading to inconsistency of dB state.

4. durability: once transaction completed, the updates should be stored & written to disk even if a system failure occurs.

→ NULL value means data is unavailable it doesn't mean 0.
eg: subjects_taken= NULL
- means subject taken is not 0 but it is unknown

→ **data warehouse:**

- central repository where data flows from transactional systems & other relational dB.
- it is process of collecting, extracting, transforming data from multiple sources & storing them in 1dB called as DW.

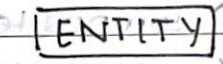


→ **data abstraction in DB:**

1. physical level
2. logical level / conceptual
3. External / View Level.

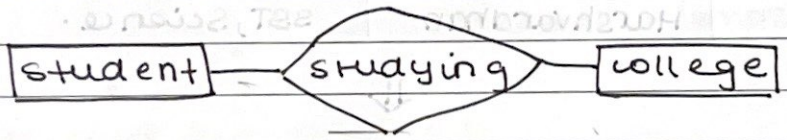
→ **Entity:** real world object.

- represented in a rectangle.



→ **Relation:** relationship btw 2 entities.

- represented in a diamond.



→ **types of relation:**

1.) one-to-one: person → has ← aadhaar card.

2.) one to many: person 1 has n friends

3.) many to many: student n has n subjects.

→ delete command (imP):
- removes rows based on a condition provided by user using 'WHERE' clause.
- reversible.

→ lock: a mechanism to protect a shared piece of data from getting updated by 2 or more dB users at same time.
types: shared & exclusive.

→ Normalisation: reducing data redundancy by organising data into multiple tables.
- better usage of disk
- easier to maintain

1.) FIRST NF (1NF): free from multivalued or composite (derived) attributes.
- each value should be atomic (single).

stud-name	sub-taken
Shivangi	Maths, science
Harshvardhan	SST, Science.

⇓

stud_name	subj_taken
shivangi	maths
Shivangi	science
Haushvardhan	SST
haushvardhan	Science

→ Second Normal form (2NF):

- provided that data is in 1NF.
- it shouldnt have any partial dependencies which means an attribute should depend only on a part of primary key & not on whole key.

stud_ID	sub_ID	marks	teacher
191	13A	70	Java
192	13B	75	python
193	13A	80	Java

↙ ↘

stud_ID	sub_ID	Marks
191	13A	70
192	13B	75
193	13A	80

sub_ID	Teacher
13A	Java
13B	python

→ Third Normal form (3NF):

- should be in 2NF
- no transitive dependency.

eg: DB has emp_id, emp_name, emp_zipcode, emp_state

↙ ↘

Emp_id	emp_name	empzipcode

empzip	emp state

→ Boycee codd normal form!

- for every dependendency of A on B (A → B) A should be superkey of table.

eg: Emp_ID, Emp_addr, course_ID, course_name.

table 1: {Emp_ID, Emp_addr}

table 2: {course_ID, course_name}

table 3: {emp_ID, course_ID}

candidate key
- set of keys that uniquely identify.
- 1 of these is primary key

super key
defines a set of attributes that can uniquely identify

Types of keys → composite.

primary	unique	alternate	foreign
- uniquely identifies a row	- same as primary but can take null values.	candidate keys not chosen as primary key	joins two tables & is common to both.

→ **Entity types:**

① **weak**: is not sufficient to become a primary key.
- enclosed with ~~for~~ double diamond / rectangle.

② **strong**: strong to become a primary key.
- single diamond / rectangle.

Camlin Page
Date / /

SQL

→ **Primary key:**

```
create table students
( ID INT NOT NULL,
  Name varchar (20),
  primary key (ID)
);
```

→ **Unique key:**

```
create table students
( ID INT NOT NULL unique
  Name varchar (255)
  unique (ID)
);
```

→ **foreign key:** FK — references — ( )

```
create table students (
ID INT NOT NULL PRIMARY KEY
Name varchar (255)
Library_ID INT
FOREIGN KEY (LIBRARY_ID) references library (library ID)
);
```

(table 2 name)

→ **check:** allows certain data only.

```
create table Persons
( ID INT NOT NULL,
  Name varchar (255),
  Age INT,
  check (Age >= 18)
);
```

→ **where:** filter records that are necessary based on specific conditions.

```
select *
from student_table
where graduation_year = 2019;
```

→ **order by:** sorts records in ascending order by default.

```
select *
from student_table
where graduation_year = 2019
order by stud_ID;
```

→ **having / group by:** both used together w/ aggregate funcs.

→ select dept_ID, Avg (salary)
from employees,
group by dept_ID;

→ select dept ID, avg (salary)
from employees,
having avg (salary) > 8000;
group by dept id;

→ **Aggregate functions:**
- avg, count, max, min, sum.

```
select count(*)
from employees,
where salary = 10000;
```

→ String function:

1.] LIKE: select *
from students
where Fname LIKE 'SH%';

2.] %: represents 0 or more characters.

select max(salary)
from employees
where job-ID LIKE '%REP%';

3.] _ : represents one character only.

select * from employees
where Fname like _hivangi;

→ Set operators:

1.] union: gives all data w/o duplication from two different tables.

select FName, Lname
from student UNION
select FN, LN
from staff.

2.] union all: gives all data w/ duplication.

3.] INTERSECT: common data of 2 different tables.

4.] MINUS: removes common elements b/w table 1 and table 2 from table1.

→ IN command : should be from given data set.

```
select *
from employees
where state IN ('Goa', 'Delhi' 'Maharashtra');
```

→ **Alias**: a temporary name assigned to the table column.
- also good if we want to secure the real names of database fields.

→ a must.

```
select max_salary AS bonus
from employees;
```

→ **Views**: a particular view of table based on conditions.

```
create view stud_view1
as select name, RNO
from student,
where marks >= 180;
```

→ **update**: make changes in original data:

```
update stud_view1
set marks = 150
where marks <= 130;
```

→ **Joins**: done b/w tables having 1 common table.

1.) **Inner / Equi Join**: same data present in both the tables.

```
Select E_ID, E_name, E.dept, Dept.dname
from employees E
inner join dept
on E.dept = dept.dname;
```

2.) **left outer join**: retrieves all from Left and matched ones from right.

```
Select *
from table_A left join table_B
on A.col = B.col;
```

3.) right outer join: all from right and matched ones from left.

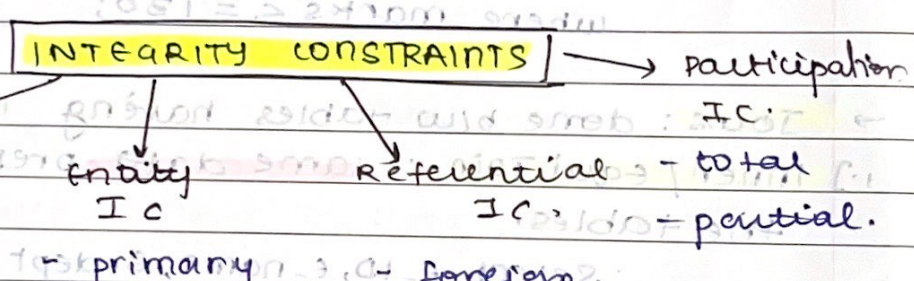
4.) full outer join: matching data from left or right.

5) self join:
- table alias used.

```
select A.ename , B.ename
    from emp A , emp B
    where A.manager = B.job_ID;
```

6.) cross join: cross product of both tables (all data will be printed).

```
select s.name, is.sub
    from students as S
        cross join subjects as is;
```



→ DBA (database administrator):
- selects HW & SW
- improves query performance
- takes care of DB.