14/03/2023

DATABASE MANAGEMENT SYSTEMS

* Data: collection of raw values.
  ↳ plural word
  Datum → singular word.
* Processing the data: Information
* DBMS helps (or) acts as interface to convert data to information.

100 200
150 90
raw data

	Bottles	drinks
2020-21	100	200
21-22	150	90

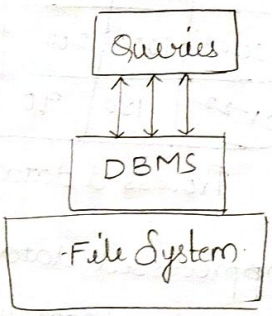
Processed data: information

* Information (or) processed data helps in extracting knowledge from the database.
* Hence; we can take decisions based on knowledge.
* Data storing & accessing from files (or) different memories in a system; which is quite difficult to get information; but this is easier while using DBMS.
* DBMS creates a data model in the

form of tables; where data is extracted and stored in a systematic form.

DBMS → Logical representation
FILE storage → Physical representation

*→ Any question required to be solved using a stored data in a DBMS are called QUERIES.



DBMS is implemented over a file management system.

*→ Database is a collection of interrelated information / data.

→ The fields of database depends on the requirement of the situation.

→ Database management deals with storing, accessing, retrieving and modifying the data.

→ Presentation view: User interacting with the DBMS queries.

* SQL - Structured Query Language

* Data → Structured data
→ Unstructured data.

→ Oracle: Relational DBMS where data is structured.

*→ SQL → Procedural SQL → We can specify what data to be extracted & how to extract
→ Non-Procedural SQL → We can specify the data to be extracted

Tuple Relational Database
Relational Algebra
Domain Relational Calculus
} Procedural SQL

SQL → Non-Procedural SQL

* Process of getting knowledge from the database is called DATAMINING.

* ER Model: pictorial representation of data

A – Atomicity : all operations are executed (or) not
C – Consistency: total amount before & after transaction
I – Isolation:
D – Durability:

When simultaneous transactions are taking place at the same time which are called concurrent transactions (using same data object in more than one transaction); then the transaction locks that data object and the other transaction must wait for a less time which can be done using ISOLATION property.

*→ Durability deals with having an updated value in the database after any transaction.

*→ updated value is done using comit () function

→ These ACID properties helps to have a successful transaction.

→ Same data present in different tables is REDUNDANCY.

→ Dividing big tables into smaller ones is called DECOMPOSITION.

→ Normalization techniques are used to reduce redundancy.

16/03/2023

UNIT-1 : INTRODUCTION

RDBMS – Oracle, MongoDB

→ DBMS contains information about a particular enterprise:

* Collection of interrelated data
* set of programs to access the data.
* an environment that is both convenient & efficient to use.

→ Applications: → very large

1) Bank transactions
2) Airline reservations, schedules
3) Universities – student details
4) Sales – purchases
5) Online shopping
6) Manufacturing Industries.
7) Employee records, tax deductions.

1TB = 1024 GB
1PB = 1024 TB

DBMS started in early 80s.

→ Every OS has file processing systems which handles files in a system.

→ Drawbacks of file system: / Advantages of DBMS

* Data Redundancy & inconsistency.
* Difficult to access data.
* Data isolation
* Integrity problems – hard to add or change the constraints in Files. → we need to write a code to put constraint to a code.
* Query processing engine → converts the query into a code and solves the problem

* While creating a table; every attribute has a specified data type.

Eg: We cannot assign real no. to an integer datatype → domain constraint.

* Types of Constraints:

→ Check
→ Not NULL
→ Unique
→ Referential → Primary key, Foreign key

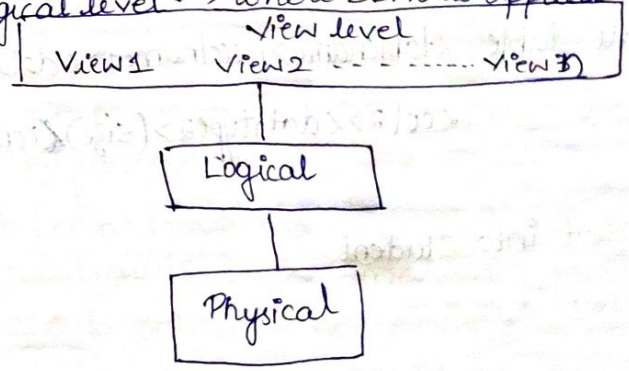
* Atomicity: It must execute all the instructions (or) should not execute any of them.
→ File processing system cannot check for atomicity properly; whereas this can be done easily by DBMS

* Simultaneously no. of users cannot access data in the files whereas DBMS supports concurrent access by multiple users.

* Security is not provided by files while accessing data.

→ Levels of Abstraction

* View level → just writing queries
* physical level → where the actual data is stored physically.
* logical level → where DBMS is applied.



17/03/2023

* SQL CONSTRAINTS:

→ Limit the type of data that can be inserted into the table.

→ ensures accuracy & reliability of data.

→ Constrainsts can be columnlevel or table level.

1) NOT NULL:

→ by default; column can hold NULL values

* enforces a column to NOT accept NULL values; means this constraint asks the user to enter data to this column before updating or adding a record.

Syntax:

```
create table <tablename>(<colname1><datatype1>(size)>
        <col2><datatype2>(size)<constraint>);
```

→ ~~insert into student~~

Ex:

```
create table Student (rollno varchar2(15),
                      name char (20) NOT NULL,
                      class varchar2(5));

insert into Student values('052', 'Sivani',
                           'CSE');
// 1 row created

insert into student (rollno, class)
                     values('052', 'CSE');
// Error

insert into student (rollno, name)
                     values('041', 'Sarayu');
// 2 row created
```

2) UNIQUE:

→ ensures that all values in the column are different

→ even primary key constraint helps for uniqueness.

→ We can have more than one unique constraints but primary key constraint should only be one per table.

* Ex:

```
create table Student (rno varchar2(15),
                      name char(20) NOT NULL,
// column level        unique(rno));

create table ze (a varchar2(15),
                 b char(20) NOT NULL,
// table level      c Numeric, unique(a,b));
```

→ if we insert same rno, we get error.

* NULL is not a unique value.

* if we don't give rno in 1st round, rno = NULL

* in second round also we do not give rno.
  rno = NULL
  ⇒ no error.

3) PRIMARY KEY

→ uniquely identifies each record in a table.

→ must contain UNIQUE values and cannot contain NULL values.

→ table can have only one primary key but this primary key can contain single or multiple columns.

Ex:

→ If the combination of name & class is primary key then 2 names can be same; if 2 classes are different.

↳ primary key is used to differentiate one record from another.

```
* create table Student (name varchar2(20),
                        class varchar2(10),
                        rno varchar2(15),
// column level                PRIMARY KEY);

* create table student(dno varchar2(3),
                       dname char(10),
                       loc char(10),
                       PRIMARY KEY((dno)));
```

4) FOREIGN KEY:

→ Insert values in table 1 then to table 2; and values of foreign key shd match primary key

→ prevent actions that would destroy links between tables.

→ is a field (or collection of fields) in one table; that refers to the primary key in another table.

→ table with foreign key: child table (secondary)

table with primary key: parent table (primary)

→ prevents invalid data from being inserted into foreign key column; because it has to be one of the values contained in parent table.

PID	Name	Age
1	Ola	30
2	Tove	23
3	Kari	20

OD	ONumber	PID
1	77895	3
2	44678	3
3	22456	2

```
create table Orders
( OD Number(10,0) NOT NULL primarykey,
ONumber Number(10,0) NOT NULL,
PID Number(10,0) FOREIGN KEY
REFERENCES Persons(PersonID));
```

5) CHECK:

→ used to limit the value range that can be placed in column.

→ if we define CHECK constraint on a column it allows certain values for this column.

→ if we define CHECK constraint on a table; it allows the values in certain columns based on values in other columns in the row.

Ex:
```
create table <tname>
(<cname><dt>(<size>)check(condition));
```

```
create table Person (~~ID NOT NULL~~
(ID Number(10,0) NOT NULL,
Name varchar2(10),
Age Number(3,0) CHECK(Age>=18));
```

6) DEFAULT:

→ used to set a default value for a column

→ added to all new records; if no other value is specified

Ex:
```
create table Persons
(ID Number(10,0) NOT NULL,
Name varchar2(15),
City varchar2(10) DEFAULT 'Hyd');
```

* SCHEMA:

→ can be defined as the design of a database.

→ overall description of database is called DATABASE SCHEMA.

- → **Physical Schema:**
  defined as the design of database at its physical level.
  Expressed how data is stored in blocks of storage.
- → **Logical Schema:**
  defined as the design of the database at its logical level.
  Data can be described as certain types of data records that can be stored in the form of data structures.
- → **View Schema:**
  defined as the design of the database at view level; which describes end-user interaction with DBMS.

→ physical data independence – ability to modify the physical schema without changing logical schema.

* DATA MODELS:

→ modelling of data description, and consistency constraints of the data.

→ provides conceptual tools for describing the design of the database.

→ Four Models:

- → **Relational Data Model:** $\boxed{\text{tables – relations}}$
  - * Designs data in the form of rows & columns
  - * uses tables for representing data & in-between relationships.
  - * used by commercial data processing applications.
- → **Entity-Relationship Data Model:**
  - * logical representation of data as objects & relationships among them.
  - → objects ↔ entities; relationship ↔ association among these entities.
  - * set of attributes describe the entities.
  - * set of same type of entities is known as an "Entity Set" & set of same type of relationships is known as

"Relationship set"

→ Object-Based data model:

* extension of ER model with notions of functions, encapsulation and object entity.
* structured and collection types.
* objects are data carrying its properties

→ Semi-Structured Data Model:

* allows the data specifications at places where the individual data items of the same type may have different attribute sets.

* relationship: ~~relation~~ association b/w 2 (or more relations/tables.
→ most relations are binary.

* Database Engine: responsible to store, manipulate & retrieve data.

*→ 3 components:

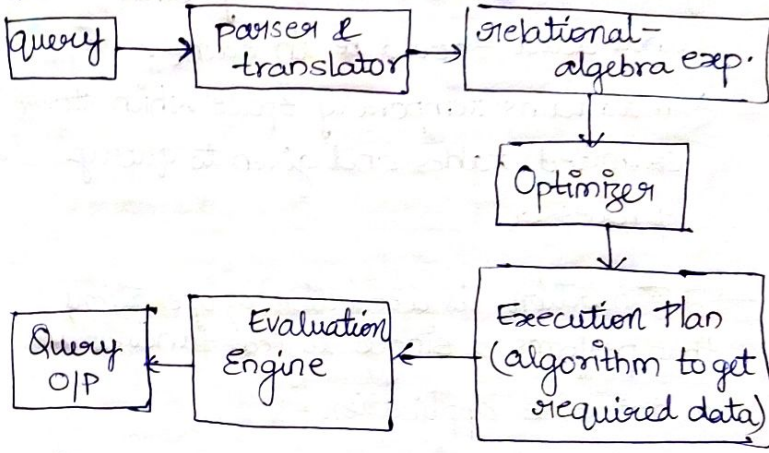
→ Storage Manager:

*→ interacts with the file system at an OS level to store data.
* interface b/w query processing system and database.

* Query processing

→ changes query to relational ~~database~~ algebra expression.

*→



→ Steps:

→ Parsing & Translation
→ Optimization
→ Evaluation.

*→ Storage Manager converts DML into commands of file system.

*→ 4 components:

→ Authorization & integrity: check for ability to access privileges and constraints
* Integrity of data deals with checking for

each operation to execute completely or not.

→ File Manager: allocates space in disk for newly inserted data.

→ Buffer Manager: manages transfer of data from disc to main memory and what data should be in cache.

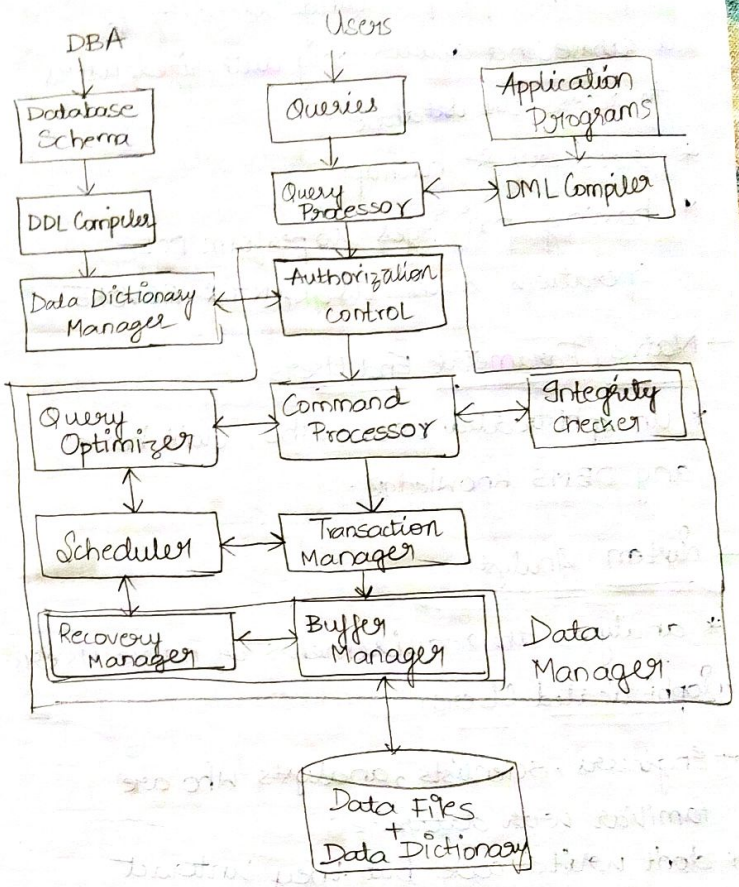
* maintains temporary space which stores extracted tables and given to query processor.

→ A transaction is a collection of operations that performs a single logical function in a database application.

* Concurrency Control Manager control the interaction among the concurrent transactions, to ensure consistency of the database.

* Database System Structure:

- → Query Processor
- → Storage Manager
- → Disk Storage



* Types of Database Users:

→ Database Administrator (DBA):

* person / team who defines the schema & controls the 3 levels of database.

* responsible for providing security to database and allows only authorized users to access the database.
* recovery & backup.
* having privileges to perform DCL operations such as GRANT & REVOKE.

→ Naive / Parametric End Users:

* unsophisticated users who don't have any DBMS knowledge.

→ System Analyst:

* analyses the requirements of naive users

→ Sophisticated Users:

* Engineers, scientists, analysts who are familiar with database.
* don't write code but they interact the database by writing SQL queries through the query processor.

→ Database Designers:

* design the structure of database which includes tables, views etc.
* decides the type of data to be stored in the table.

→ Application Programmers:

* system analysts – backend programmers who writes code for application programs

→ Casual Users:

* occassionally use the DBMS & require new information.

*→ Database drivers are used to convert ER model to any other data model.

23/03/2023

* ENTITY RELATIONSHIP MODEL:

→ pictorial representation of a database.

*→ specify all the aspects used in the database.

→ Design Phases:

* initial phase: characterize the data needs of the given application.
* choosing a data model; translates these requirements into a conceptual schema

* users describe the kinds of operations that will be performed on the data.

*→ The process of moving abstract data model into a fully developed database in 2 phases: Logical design & physical design.

*→ ER model:

E → Entity : thing/object described by set of attributes.

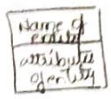
R → Relationship: association among several entities.

→ Database Modelling:

* used in mapping meanings & interactions of real-applications onto a conceptual schema.

* Basic concepts of ER Model:

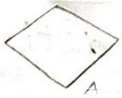
→ Entity sets
→ Relationship sets
→ attributes.
→ NOTATIONS:
→ : many to one
— : many to many
<—> : one to one



→ Strong Entity Set

→ Weak Entity Set

attribute → primary key attributes



→ Relationship Set B

If we cant give a name to the relation then we can A_B / B_A
Relation from A to B
Relation from B to A

relationship participation
= → total
— → partial participation

---- → discriminating attributes

→ Entity is an object which is distinguishable from other objects. Collection of similar entities is called entity set and they share same properties.

* Entity is represented by set of attributes which are possessed by all the entities.

Ex: instructor = (ID, name, street, salary)
course = (course_id, title, credits).

* Subset of attributes form a primary key of the entity set i.e. uniquely identifying each entity of the set.

* Keys:

→ Candidate Key: Set of attributes which are used to distinguish one record from other.

→ Super Key: is a candidate key whose proper subsets are not candidate keys.

* Relationship Sets:

→ Relationship is an association among several entities.

* Relationship set is a mathematical relation among $n \geq 2$ entities; each taken from entity sets:

$$\{(e_1, e_2, \ldots e_n) \mid e_1 \in E_1, e_2 \in E_2, \ldots e_n \in E_n\}$$

where $(e_1, e_2, \ldots e_n)$ → relationship.

28/03/23 (LAB.)

→ Aggregate functions: generally performed on numerical values.

* performed on columns referred to as attribute

→ Sum: sums up all the values of a particular attribute

→ avg: avg of all the values of an attribute.

→ min & max: returns min value (and) max value of a list of records.

→ count: gives a count acc. to the query/condition.

Ex: select count(empno) from emp where sal > 5000;

→ execution begins from rightside for any select statement

→ select max(sal) from emp;

→ select min(sal) from emp where deptno = 10

→ select avg(sal) from emp;

* "Group by" can be used along with aggregate functions.
  ↳ used to group a set of records according to one of the attribute
  → ~~primary key~~

→ select deptno, max(sal) from emp
Group by deptno;

O/P:

dept	max(sal)
10	2000
20	5000
30	4000

* If we apply Group by ~~option~~ function w.r.to any primary key; it gives each record with a diff. value only.

→ We must check the attribute which has same value for more than one record.

* Select deptno, count(*) from emp group by deptno
  ↓ all the records

* Select deptno, count(*) from emp group by deptno order by deptno

* Select sum(sal) from emp;

* select mgr, sum(sal) from emp group by mgr;

* select mgr, sum(sal) from emp where sal>1000 group by mgr.

→ where → group by → order by → attributes.

* Select sqrt(25) from dual;

* select mod(25,4) from dual;

* select 23+34 from dual;
  select 34-23 from dual;

* O/P of select statement is a set.

→ String functions:

1) upper : select(upper('abc')) from dual;
2) lower : select(lower('ABC')) from dual;
3) initcap : select(initcap('abc xyz')) from dual; → 1st letter of every word becomes capital
4) instr : select instr('abcdef','c') from dual; → no. of times a specified character is given
5) substr : select substr('abcdef',2,3) from dual; → starting index, last index
6) Ltrim : select ltrim('abc') from dual; → left empty space removed
7) Rtrim : select rtrim('abc') from dual; → right empty space removed
8) trim : select trim('abc') from dual; → all extra white spaces are removed
9) ASCII → gives ascii value of a character

10) char → gives character acc. to ascii value

11) lpad : select lpad ('abc', 10, '*') from dual.
↳ length of string is left
↑sed using *.

12) rpad : select rpad('abc', 10, '*') from dual.
↳ length of string is
right ↑sed using *.

13) length : select length('abcde') from dual.

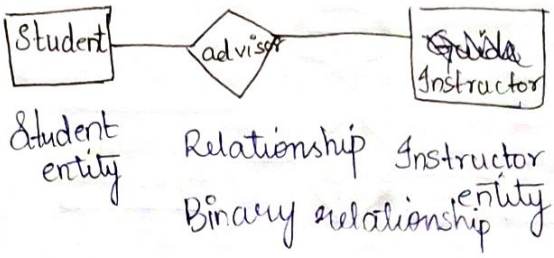
31/03/2023

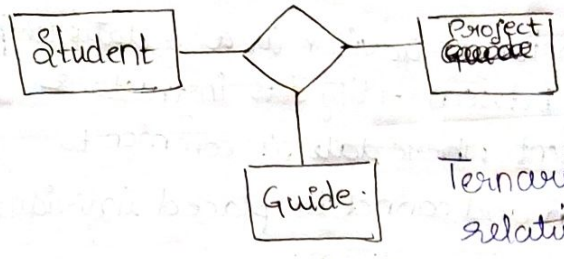
* **Strong entity Set**: Sufficient attributes to form a primary key.

* **Weak entity Set**: No sufficient attributes to form a primary key. It depends on one of the strong entity set.

* **Binary relation**: relation b/w only 2 entities.

* **Ternary relation**: relation b/w 3 entities





→ Relationship Set contains the attributes.

→ To represent any relation between 2 entities; it is better to use primary keys of both the entities.

* If relationship is one to many → take the primary key of many side

* If relationship is one-one → take any of the primary key.

* If relationship is many-many → combination of both primary key.

*

⇓

For instance, advisor is a relationship set between entity sets instructor & student where date is common to both and cannot be placed individualy in either of them.

→ **Degree of Relationship Set:**

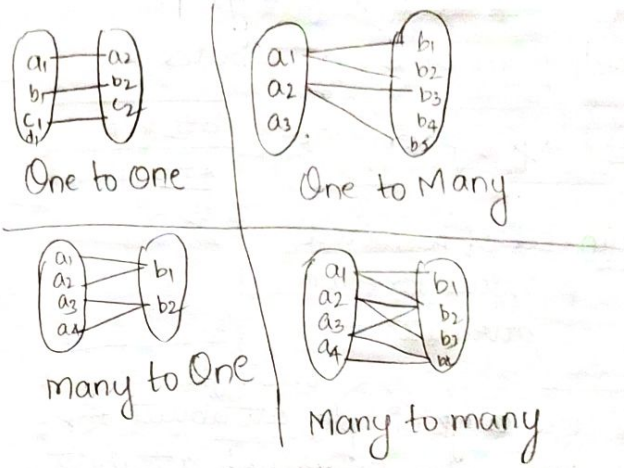
* no. of entity sets involved in certain relationship.
* Most relationships are binary.
* Relationship other than binary can be converted to binary.

→ **Mapping Cardinality:**

no. of entities to which another entity can be associated via a relationship set.

** **Types:**

One to One, One to Many, Many to one, Many to many.



* **Attributes:** → one of the features of entity (character type)

→ Simple attribute: only single value.
* represented as RNo Ex: Rno.

→ Composite attribute (character with multiple values): more than 1 value
Ex: Name, First, las

* represented as

Address
- Street
- HNo
- City
- State

Name
- First
- Last

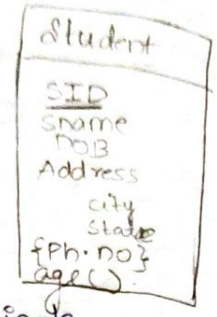
→ Single Valued: RNo (numerical value)

→ Multi Valued: {ph.no} (multiple numerical values)

* Derived attributes: age()

computed from other attributes.

Ex: age, given dob.



→ Domain: set of permitted values for each attribute.

* If we use ~~single~~ simple (or) single attributes for composite (or) multi-valued attributes we may have more NULL values.

* Redundant Attribute:

↳ repeated attributes ~~with~~ in different entity tables only if that attribute is not a foreign key in any of the table.

* Types of Joins:

↳ cartesian product: joins every record of A entity with every record B entity

→ Natural join: joins the 2 tables according to the common attribute of both tables

→ Outer join: → we need to specify the condition

Ex:

→ left outer join: we can append the record in the left table to the joined tables set where that record does not have a match in right table.

→ right outer join: appending the record in right table to the joined table set where that record does not have a match in left table.

→ full outer join: joining all the records which are not having match in the other table.

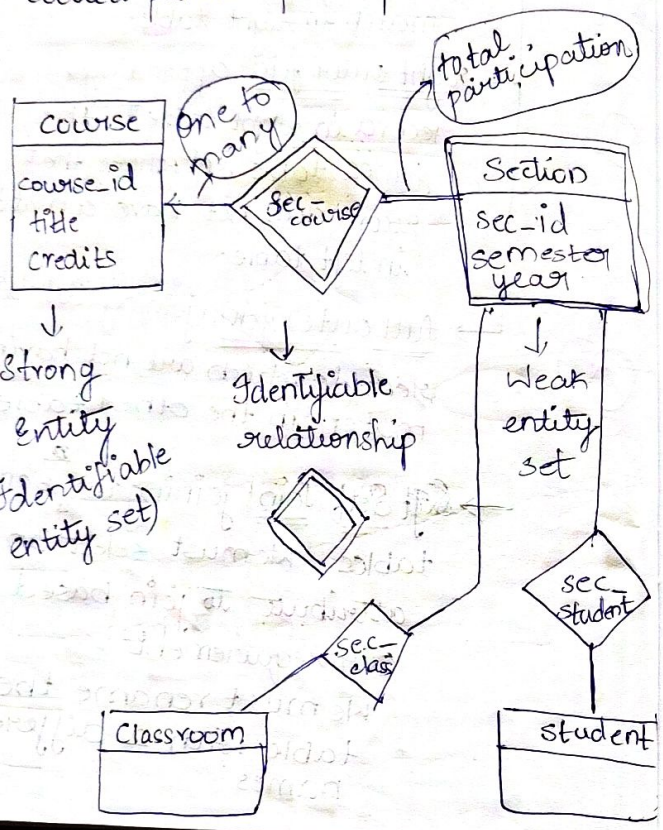
→ ~~Self~~ Self Join: joining the same tables. We must select an attribute to join based on our requirement. We must rename the table with 2 different names

04/04/2023

→ If all the entities in the set are participating in the relationship then it is called total participation

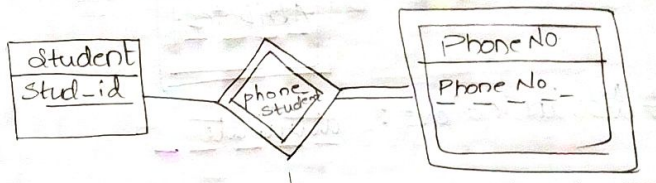
→ If few entities are not participating in the relationship then it is called partial participation.

*



→ The attributes in the weak entity which can be called discriminating attributes along with primary key of the identifiable/strong entity set can be given as primary key for weak entity set.

→ If a table is having more NULL values, table becomes sparse.

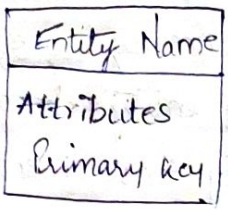


↓ multi-valued attributes with NULL values can be solved

Stud-id	Phone No
001	#1
002	#1
002	#2
003	#1

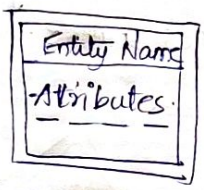
* A foreign key of an entity cannot be a primary key or a part of primary key to that table

* Representations in ER diagram

1) 

→ strong Entity set

2) primary key : attribute

3) Weak Entity Set : 

4) discriminators : attribute

5) Relationships :

6) attributes of a relationship :

7) one ← many : one to many | ~~many to one~~

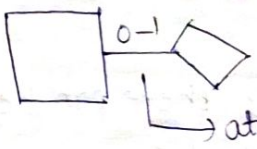
8) — : many to many

9) ⟷ : one to one.

10) ═ : total participation

11) — : partial participation.

12) → : many to one

13) 

→ atmost 1 is participating

0..* : 0 ~~one~~ to many
1..* : 1 "
1..1 : one to one

14)

→ atleast 1 is participating

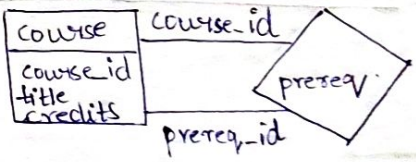
15) One to one
one to many } → total participation.

16) 0 - 1 → partial participation
if 0 is allowed

17)

→ for the attribute to be assigned to relationship.

18) Relationship between same entity sets.



* course_id and prereq_id are roles of a relationship.

* Between 2 same entity sets, there can be more than one relationship.
  ↳ Role of the entity participating in the relationship.

08/04/2023

* CONVERSION OF ER model to Relational Model:

→ A strong entity set reduces to a schema with same attributes

student (ID, name, tot_cred)

→ A weak entity set becomes a table that includes a column for the primary key of identifying entity set

section (course_id, sec_id, sem, year).

→ Relationship into a relation/table
  ↳ attributes are the primary keys of the 2 participating entity sets, and any descriptive attributes of the relationship set.

Ex: advisor(s_id, i_id)

s_id → primary & foreign key for student, foreign key for instructor

i_id → primary & foreign key for instructor, for student

* primary key of a relationship depends on type of relationship.

→ one to many relation → primary key of many side is the primary key of relationship

→ one to one relation: either of the primary key

→ many to many relation: combination of 2 primary keys.

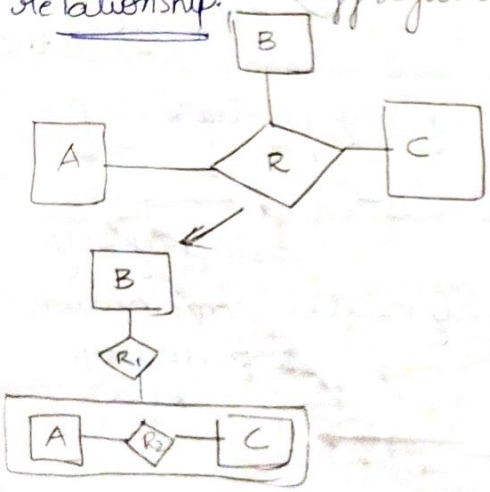
→ Composite attributes are flattened out by creating attribute for each separate attribute.

→ A multi-valued attribute M of an entity is represented as EM schema.

* Considering the attribute as a weak entity set.

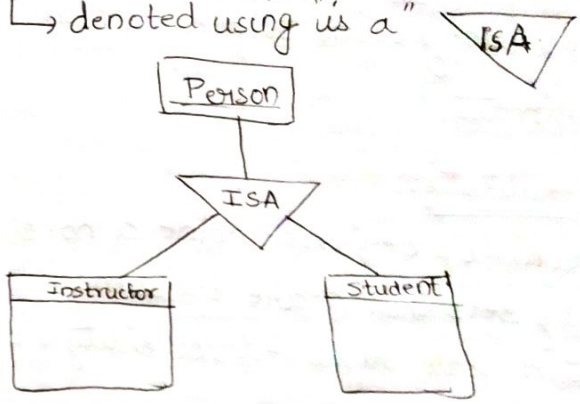
* → Conversion of a non-binary relation into a binary relationship using aggregation.

→ Cardinality Constraints on Ternary relationship: (Aggregation)



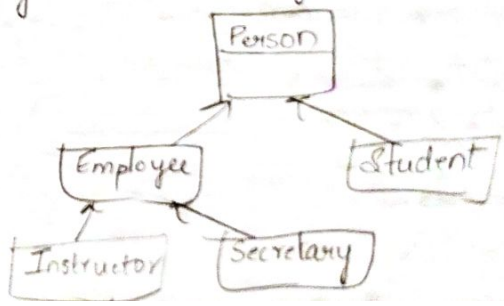
* Specialization:

→ denoted using "is a"



→ Top-Down process; sub-groupings with an entity set are distinctive ~~for~~ from other entities in the set.

* Sub-groupings become lower-level entity sets that have attributes in relationships that do not apply to the higher level entity set.

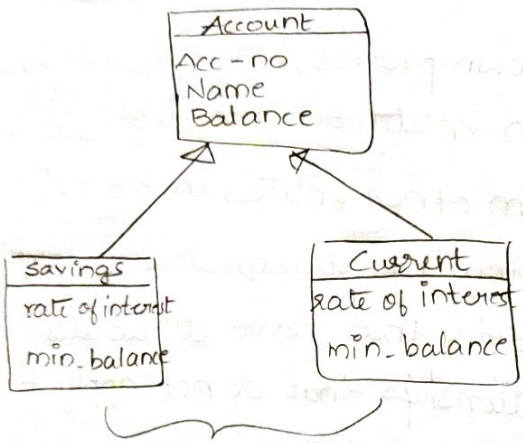


primary key

→ Attribute ~~are~~ is taken from the parent class along with the present class attributes

* Generalisation:

→ Bottom-up design: combine a no. of entity sets that share the same features into higher-level entity set



Share the common features along with generalized entity sets/features

→ Specialization: mostly b/w physical entities

→ generalization: mostly b/w logical entities

* Relations are unordered

* Keys:

A set of attributes which uniquely identifies a tuple.

ID → super key

[ID] → attribute

$t_1[ID] \neq t_2[ID]$ → tuples form disjoint.

ID, name → candidate key

$t_1[ID, name] \neq t_2[ID, name]$

→ A super keys super sets are also super keys.

* Candidate Key: Minimal Key

→ One of the candidate key is a primary key.

Single attribute candidate key is a primary key.

* Foreign Key:

Attribute which is referring to another table.

Procedural : Relational algebra

Non-Procedural : sql, tuple, relational calculus

→ Relational algebra (Procedural Query Language)
↳ 6 operations

$\pi_{ename}(\sigma_{sal > 5000}(emp))$

1) Selection (σ): In SQL equivalent class is where $\sigma A = B \wedge D > 5(r)$

A = B and D > 5 from relation r

→ Let r is a relation. Select subset of r that follows condition.

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

A	B	C	D
α	α	1	7
β	β	23	10

$\sigma_{salary > 1000}(emp)$ ⇒ select * from emp where salary > 1000

2) Projection (π): Attributes that must be extract
↳ equivalent class is select

($\pi_{empno, ename}(emp)$)

→ select empno, ename from emp;

In a relation same tuple is not repeated.

r_2

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

$\Rightarrow \pi_{A,C}(r)$

A	C
α	1
α	1
β	1
β	2

⇒

A	B
α	1
β	1
β	2

$\pi_{ename, sal}(\sigma_{sal > 1000}(emp))$ ⇒ select ename, sal from emp where sal > 1000;

3) Union (u): On 2 relations (null have same values) They must have same

structure.

r =

A	B
α	1
α	2
β	1

s =

A	B
α	2
β	3

r ∪ s ⇒

A	B
α	1
α	2
β	1
β	3

$\Pi_{ename, deptno}((\sigma_{sal} < 1000) \cup (\sigma_{sal} > 5000))$

employees salary below 1000 and above 5000.

(select * from emp where sal<1000) union (select * from emp where sal>5000);

4) set difference (r-s):

r =

A	B
α	1
α	2
β	1

s =

A	B
α	2
β	3

r − s =

(select * from emp where sal>2000) minus (select * from emp sal>5000);

$\Pi_{ename, deptno}((\sigma_{sal} > 2000) - (\sigma_{sal} < 5000))$

5) Intersection (r ∩ s) = r − (r − s)

r =

A	B
α	1
β	2
β	3

s =

A	B
α	1
β	4
α	3

r ∩ s =

A	B
α	1

6) **Rename (ρ):**

$\rho_x(E) \Rightarrow$ e is renamed as x.

7) **Cartesian Product ($r \times s$):**

r =

A	B
α	1
β	2

s =

B	D	E
α	10	a
β	10	b
β	20	b
γ	10	b

$r \times s$ =

A	r.B	s.B	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
⋮	⋮	⋮	⋮	⋮

8) **Natural join:** Subset of cartesian product

* $r \bowtie s \rightarrow$ equate the common attribute values.

* left outer join : $r ⟕$
* right outer join: $r ⟖ s$.

* → **Full outer join:**

$r ⟗ s$

σ, Π, $\times$, $\cup$, $-$, $\bowtie$

$\cup$ → union

$\bowtie$ → natural join

σ → selection

Π → projection

$\times$ → cartesian

$-$ → set difference

Sum
AVG
max
min
→ additional operations

* **Group by ($\mathcal{G}$)**

$\Pi_{deptno}(avg(sal)\ \mathcal{G}_{deptno}(emp))$

$\Pi_{ename, dname}(\sigma_{sal>5000}(emp \bowtie dept))$

Select ename, dname
from natural join emp

from dept where sal > 5000.

19/03/2023

* NESTED SUBQUERIES:

→ We can use only select statement as a sub query.

Select, from } mandatory clauses for a select statement

where, groupby, order by, having, takes } optional clauses.

* Subquery is a valid select statement

* Criteria: for subqueries: → outer query o/p

→ set membership ⇒ LHS = RHS → inner query o/p.

→ set comparisons

→ set cardinality.

* Subqueries can be done in

select $A_1, A_2 \ldots A_n$

from $r_1, r_2, \ldots r_m$.

where $W_1, W_2, \ldots W_m$.

→ Query 1 attribute in Query 2

→ Qury 1 attribute not in Query 2

* select name from instructor

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

atleast one of the value.

→ > all : > all the values

→ exists:

Select course_id from section as S

where sem = 'Fall' and year = 2009

and exists (select * from section as T

where S.course_id = T.course_id);

allows to use 'S' in inner query

* select deptname, avgsal from (select dept_name, avg(sal) as avgsal from instructor group by deptname)

where avgsal > 42000;

↳ a from clause: if

we get a virtual table, which is a temporary one.

* VIEWS:

→ Views are virtual relation of original table

→ provides a mechanism to hide certain data from the view of certain users.

* create view v as <query exp> → legal select exp

* once view is defined; the view name can be used to refer to the virtual relation that the view generates.

* create view faculty as select ID, name, dept_name from instructor;

* We can create a view from another view.

* Materialized views:
Creating a physical table from the result of the query from view.

28/04/2023

* NORMALIZATION:

→ When we combine 2 tables; without any relationship set; hence we get a repetition of information.

* A good database deals with having every other attribute depending on key attribute.

{ID} → {name, salary, dept, name}

{dept_name} → {building, budget}

table 1 : {ID, name, salary, dept_name}

table 2 : {dept_name, building, budget}

table1 natural join table2.

* If we can divide a table into 2 tables and joining them naturally gives the original table; then such decomposition is called loss-less decomposition.

* But these decompositions; shall not allow us to insert NULL values to its tables.

*→ Combined Schema: Joining 2 tables to get a good design.

→ Not all decompositions are good; where we do not get the original table.

* 1 NF do not depend on functional dependency.

*→ First Normal Form:

* Every attribute should be atomic attribute.

• Atomic: Singlevalued attributes only.

Branch (PK)	Mgr	Location
A	X	{Hyd, Vij}
B	Y	{Blore, Delhi}
C	Z	{Kol, Chenn}

Not 1NF

⟹

Branch	Mgr	Loc
A	X	Hyd
A	X	Vij
B	Y	Blore
B	Y	Delhi
C	Z	Kol
C	Z	Chenn

PK = (branch, loc)

1NF

Branch	Mgr	Loc-ID
A	x	1
A	x	2
B	y	3
B	y	4
C	z	5
C	z	6

Loc-ID	LOC
1	Hyd
2	Vij
3	Blore
4	Delhi
5	Kolkata
6	Chennai

⇓

In this case, if new location / branch is added ; so as to ~~[illegible]~~ modify easily.

* Remaining all normal forms depend on functional dependancy.

* Let R be relation schema.

$\alpha \subseteq R$; $\beta \subseteq R$

Consider 2 tuples t1 & t2

~~[illegible]~~ if $t1[\alpha] = t2[\alpha]$

then $t1[\beta] = t2[\beta]$

then $\alpha \rightarrow \beta$ is ~~a~~ a ~~relation that~~ is a functional dependency that holds good.

A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
a_2	b_2	c_3
a_3	b_2	c_4

$A \rightarrow B$ ✓

$B \rightarrow A$ ✗

$C \rightarrow A$ ✓

$C \rightarrow B$ ✓

$C \rightarrow AB$ ✓

∴ C is a primary

$A \rightarrow BC$ ✗

$AB \rightarrow C$ ✗

* $\boxed{\alpha \rightarrow \beta}$ is a trivial functional dependency if $\alpha \subseteq \beta$ $\boxed{\text{Eg: } AB \rightarrow A}$

* If $\alpha \not\subseteq \beta$; then such dependencies are non-trivial functional dependencies.

functional Dependency: $(\alpha \rightarrow \beta)$ holds on R if and only if for any 2 legal relations $r(R)$, whenever 2 tupples t_1, t_2 of r agree on the attributes α, then they also agree on β.

$$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$$

ex:

1 4
2 3 → holds
5 7

1 4
1 5 → doesnot hold. $t(1) = 4$, $t(1) = 5$
5 7

→ If $K \subseteq R$, $\forall t_1, t_2$

if $t_1[K] = t_2[K]$

$t_1[B] \neq t_2[B]$

then K is a superkey.

→ inst_dept (ID, name, salary, deptname, building, budget).

→ If K = {ID, Dept-name} → {name, salary, building, budget}.

→ To identify Super key, then.

if $t_1(K) = t_2(K)$
$t_1(B) \neq t_2(B)$ } in this case K is super key.

→ Let, R is a relational schema

F is a functional dependencies

closure set of FDs (F+) → derived dependencies

* We can find closure by following rules also known as Armstrong's Axioms:

→ If $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ (reflexivity)

If $\alpha \rightarrow \beta$, then $\gamma\alpha \rightarrow \gamma\beta$ (augmentation)

If $\alpha \rightarrow \beta$, $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ (transitivity)

Additional rules:

Union Rule: $\{\alpha \rightarrow \beta, \alpha \rightarrow \gamma\} \Rightarrow \{\alpha \rightarrow \beta\gamma\}$

Decomposition: $\{\alpha \rightarrow \beta\gamma\} \Rightarrow \{\alpha \rightarrow \beta, \alpha \rightarrow \gamma\}$

pseudo transitivity: $\{\alpha \rightarrow \beta, \gamma\beta \rightarrow \delta\} \Rightarrow \{\alpha\gamma \rightarrow \delta\}$

** ex: R = (A, B, C, G, H, I)

F = { A → B
A → C
CG → H
CG → I
B → H }

from {A → B, B → H} ⇒ {A → H} [∵ transitivity]

and

by augumenting A → C with G

AG → CG.

By transitivity {AG → CG, CG → I} ⇒ {AG → I}

AG → I.

and,

{CG → H; CG → I} ⇒ {CG → HI}

(or)

by augumenting CG on CG → I we get

CG → CGI

(CG → H) × I → CGI → HI

then,

$\boxed{CG \rightarrow HI}$

Algorithm:

Initially $F^+ = F$.

repeat

for each dependency f in F^+

apply reflexivity & augumentation on f

add that f to F^+

for each pair f_1, f_2 in F^+

if f_1, f_2 can be combined by transitivity

then add resulting to F^+

until F^+ doesnot change any other.

Closure of Attribute Set:

let, R, F be Relation schema and set of functional dependencies

$\alpha \subseteq R$.

$(\alpha)^+ \rightarrow$ closure of α.

To find $(\alpha)^+$:

result := α;

while (changes to result) do

for each $\beta \rightarrow \gamma$ in F do

begin

if $\beta \subseteq$ result then result := result $\cup \gamma$

end

$\rightarrow$ $(\alpha)^+$ gives us candidate key

ex: $R = (A, B, C, G, H, I)$

let closure = $(AG)^+$

result = AG.

$A \rightarrow B$

$B \rightarrow C$

result = AGB

$A \rightarrow C$

result = AGBC

$CG \rightarrow H$

result := AGBCH

$CG \rightarrow I$

result := AGBCHI

$(AG)^+$ = AGBCHI

~~either $(A)^+$ or $(G)^+$ is a super key or not?~~

~~$\rightarrow$ If A^+ or G^+, either of one is not a superkey then AG^+ is candidate key~~

→ For AG to become a candidate key, A, G should not be a super key i.e

→ A, G should not contain all the attributes.

→ If a Relation schema is updated, then it should also satisfy the ~~updated~~ all the ~~scher~~ functional dependencies.

Uses of Attribute Closures —

Canonical Cover of FDs (F_c): It is a algorithm in which If some set of functional dependencies holds then all the functional dependencies holds even though relational schema is updated.

Uses of

→ If relational schema, FD's are known then if $(\alpha)^+ = R$, then α is a superkey.

→ If non-of the subset of α are superkey then α is candidate key.

→ Canonical Cover: $F_c \subseteq F$.

→ If there is any extra attribute 'A' which doesnot effect the functional dependencies then we remove such attribute.

$\alpha \rightarrow \beta$

$A \subseteq \alpha$ or $A \subseteq \beta$

→ No need to include this extroneous attributes in canonical cover.

Algorithm:

$F_c = F$

repeat

apply union rule update F_c

for any $\alpha \rightarrow \beta$. Let A be any attribute

if $A \subseteq \alpha$ or $A \subseteq \beta$ is an extraneous attribute

then $F_c = F_c - \{\alpha \rightarrow \beta\}$

until (no change in F_c)

Finding extraneous attribute:

let R → Relation schema, F → set of functional dependency

method 1: $\alpha \rightarrow \beta$ → F.D

A is an attribute. $A \subseteq \alpha$.

$\gamma = \alpha - \{A\}$

$(\gamma)^+$ from F

if $(\gamma)^+$ includes β then A is extraneous

$BC \rightarrow A$

Checking B

$BC - B = C$

if → $C^+ \rightarrow A$ then B is extraneous

method 2:

$A \subseteq B$

$F' = \{F - \{\alpha \to \beta\}\} \cup \{\alpha \to (\beta - A)\}$

find $(\alpha)^+$ from F' if $(\alpha)^+$ includes A

example:

$R(A, B, C)$, $F = \{A \to BC, B \to C, A \to B, AB \to C\}$.

By applying union, $A \to BC$, $A \to B$

$A \to BC$ ($\because$ no repetition)

F becomes

$F = \{A \to BC, B \to C, AB \to C\}$

$AB \to C$

Checking if 'A' is extraneous or not

$AB - A = B$ $\left[\gamma = \alpha - \{A\} = \{AB\} - \{A\} = \{B\} \right]$

$B \to C$

Find B^+, $\gamma = \{B\}$

result $+ = B$, $B \to C$

result $+ = \{BC\}$

then A is extraneous attribute //

Now, let us take, $A \to BC$

check whether C is extraneous or not.

$F' = \{B \to C, AB \to C\} \cup \{A \to B\}$

$F' = \{A \to B, B \to C, AB \to C\}$

$(A)^+$ from F'

result $= A$

$\therefore A \to B$

result $= AB$

$B \to C$

result $= ABC$

$\therefore$ C is extraneous

$F_c = \{B \to C, A \to B\}$

lossless join:

let R_1, R_2, R be
(R is decomposed to R_1, R_2)
relational schemas

$R_1 \bowtie R_2 \equiv R$

$\to R_1 \cap R_2 \to R_1$

$R_2 \cap R_1 \to R_2$

(either R_1, R_2 intersection attributes, if it is a superkey ~~should be the superkey~~ then the decomposition is lossless join

example:

A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
a_2	b_1	c_1
a_2	b_1	c_3

(i) find functional dependencies?

(ii) find superkey.

(iii) find F_c

(i)

$A \to B$ holds

$C \to B$ holds

$AC \to B$ holds

(ii)

→ A, B, C → values are repeated. So, they are not super keys

$(AB)^+$

result += AB

$A \to B$

result + = AB → changes are not made so stop.

↓

All attributes are not included. So, not a superkey

$(BC)^+$

result += BC

$A \to B$

result += BC A is not subset of BC ✗

$(AC)^+$

result + = AC

$AC \to B$

result + = ABC ✓ superkey.

Since, A, C are not super key.

∴ (AC) is candidate key also

(iii) $F = \{A \to B, C \to B, AC \to B\}$

$AC \to B$

$\alpha \to B$

checking whether A is extraneous or not

$\gamma = AC - A = C$

$(\gamma)^+ = (C)^+ = CB$

∴ A is extraneous

$F_C = \{A \to B, C \to B\}$

Normalization → 14th chapter Normalization

Fundamentals of Database Systems

Guidelines to design Relational Schema

1. Each tuple should only contain one entities information.

COMPANY database

employee(ename, ssn, Bdate, Address, Dnumb

Department(Dname, Dnumber, Dmgr-ssn)

Dept-locations (Dnumber, Dlocation)

Project (Pname, P-number, Plocation, Dnum

works-on(ssn, Pnumber, Hours)

If G.L-1 is not followed.

→ Problems:

1) Wastage of storage

2) causes problems with update Anomalies

→ Insertion

→ Deletion

→ Modification

2) Above Anomolis should not be present

3) Relations should be designed such that their tuples will have very few NULL values.

ex: If their are any NULL values with employee name. and Count(ename) will not count that null value.

(4) we will try to reduce spurious tuples

if same names, then repeatation & wrong data

Prime Attributes: The attributes which can form a candidate key.

Non-Prime: Cannot form a candidate key.

→ If a relational schema has more than 1 candidate key, then we consid anyone of the candidate key as a primary key.

Second Normal form:-

→ An attribute that is member of the primary key K. is called a Prime Attribute

→ Full functional dependency:- a FD $y \to z$ where removal of any attribute from Y means the FD doesnot hold any more.

ex:

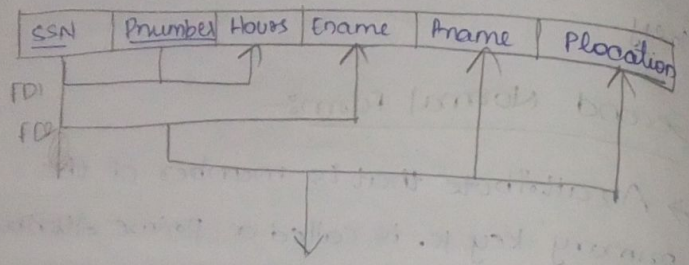
→ {SSN, Pnumber} → Hours is a full FD since neither SSN → hours nor Pnumber → hours hold.

→ {SSN, Pnumber} → Ename is not a full FD (it is call partial depend-ency. since SSN → ename also holds

→ A relation schema R is in 1NF then, it is said to be 2NF if every non-prim attribute A in R is fully functionally

dependent on primary key

= emp_proj:



EP1

SSN	Pnumber	Hours

FD1

EP2

SSN	Ename

FD2

EP3.

Pnumber	Pname	Plocation

FD3

Oracle PL/SQL

→ It is procedural oriented programming language.

→ not case sensitive

→ 2 types of functions

1. function
   ↓
   It can return a value

2. Procedure
   ↓
   → cannot return a value.

Features :-

→ Portability

→ Productivity

→ Block of statements sent as single statement

Different sections.

* Declaration section:-

→ optional

→ variables are declared here

* Executable or begin section:-

→ same as main function.

→ All the statements are written here (cod

* Exception section:-

→ traps the error which causes program to terminate abruptly.

→ we don't know whether the data is present or not.

Declare

Variables used in this program unit are decl -ed he

BEGIN

executable sections containing PL/SQL & SQL statements

EXCEPTION

statements for dealing with errors

END;

/

Datatypes

→ char, varchar2, Number, BINARY_INTEGER,

RAW, DATE, BOOLEAN,

→ LONG, LONG RAW, LOB

Variable Constraints :-

* Not Null. → cannot be empty

* constant

↳ cannot be changed

→ := → assignment operator

Predefined Exceptions

→ IN_VALID_NUMBER → no need to declare automatically raised

→ NO_DATA_FOUND

→ NOT_LOGGED_ON

→ TOO_MANY_ROWS

Syntax

Exception

when [exception name] then

action to take...

- - - -

when others then

action

END;

Conditional structures

→ If - then → If [T/F condit^n] then

statements;

END IF;

→ If - then - else → If

- - - -

ELSE

- - - -

END IF;

→ If - then - elsif.

↓

IF [ ] THEN

- - -

ELSIF [ ] THEN

- - -

ELSE

- - -

END IF;

→ Printf :- DBMS_OUTPUT.PUT_LINE

Iterative Structures :

→ LOOP ... EXIT ... END LOOP

→ LOOP ... EXIT WHEN ... END LOOP

→ WHILE ... LOOP ... END LOOP

→ FOR ... IN ... END LOOP

Program to check whether number is even or Odd :-

```
DECLARE
num number(5);
rem number;
BEGIN
num = &num;
rem = mod(num,2);
if rem=0
then
dbms_output.put_line("Number" || num || "is even");
else
dbms_output.put_line("Number" || num || "is odd");
end if;
end;
```

→ To run a program sql consists of editor.

→ ed even.sql (close the notepad)

→ @even.sql.

enter value for num: 10 } → It will not give o/p.
↓
By default

→ o/p: set serveroutput on } It executes.

@even.sql

Enter value for num: 10.

old 5 : num := #

new 5 : num := 10;

Number 10 is Even.

Procedure successfully completed.

---

3) Find sum of digits.

```
DECLARE.
num number(5);
rem number(5);
sum number(5)=0;
num1 number(5);
BEGIN
num = &num;
num1 := num;
```

```
while (num>0)
  loop
  rem := mod(sum,10);
  sum = sum + rem;
  num:= trunc((num/10);
  end loop;
  dbms_output.put_line("sum of digits of"
end;                    ||num1||"Is"||sum);
/
```

BCNF

→ A relation schema R is in BCNF if whenever an FD $X \rightarrow A$ holds in R, then X is a superkey of R

→ Each normal form is strictly stronger than the previous one.

Every 2NF relation is in 1NF
" 3NF " " " 2NF
" BCNF " " " 3NF

10/05/2023

* TRANSACTIONS:

→ Operations performed on Database are called transactions.

* ACID properties must be satisfied when a transaction is done.

* Transaction is a unit of program execution that accesses and possibly updates various data items.

* All the operations combinely perform one function.

ex: To send 100₹ from Account A to B

→ Atomicity ensures that either all the operations are executed or none is executed (rolled back).

* read: access the data from main memory
write: update the data in "

→ Consistency: ensures that the total amount or value before and after transaction must remain same.

* Commit() operation is executed to update values in main memory.

→ **Isolation**: although multiple transactions may execute simultaneously/concurrently; but the transactions must not know about the another transaction; ensuring isolated transaction.

* If there is no concurrent execution of transactions; we shall execute them serially which will not have isolation problem but the disadvantage is that; the processes must wait for the CPU.

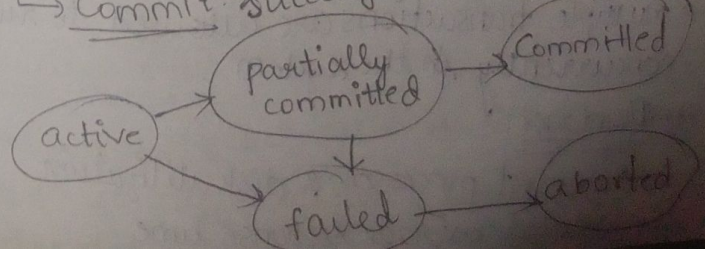
(Serial - one transaction after another)

Serializability: Taking advantage of serial execution; we execute transactions parallelly;

→ **Durability**: After a transaction is completed successfully; the changes it has made to the database must persist; even if there are system failures.

* **Storage Structures**:
  - → Volatile :- Main Memory (RAM) → if system fails (temporary data)
  - → Non - Volatile - Secondary Memory (Hard Discs) (permanent data storage; only if disk does not fails)
  - → Stable :- data storage which does not have failure and can be retrieved at any time.

* **Transaction State**:
  - → **Active**: initial state/execution state.
  - → **Partially committed**: after final statement is executed
  - → **Failed**: normal execution can no longer proceed.
  - → **Aborted**: transaction is rolled back
  - → **Commit**: successful execution



13/05/2023

→ LOCKS (ISOLATION)

= * Locks are used when more than one concurrent operations to ensure isolation.

→ Lock based protocol:
- → Exclusive Lock
- → Shared Lock

↳ 2-phase locking **

data item can read as well as write

*→ Exclusive Lock: Other transactions cannot access.

(x) compatible mode S

	S	X
	True	False
	False	False

data item can only read (S) other forms are incompatible

*→ Shared Lock: Other transactions can access but cannot modify.

→ Lock requests are made to the concurrency control manager by the programmer.

→ ACID properties are ensured for every transaction by the transaction manager else the transaction is killed/stopped.

* CONCURRENT EXECUTIONS:

→ Multiple transactions are allowed to run concurrently in the system.

*→ Advantages

→ Increased processor & disk utilization

→ Reduced average response time

→ concurrent executions increase the overhead of transaction manager.

→ Concurrency control schemes help to prevent the destroying of consistency of database.

*Schedules: a sequence of instructions that specify the chronological order in which instructions of concurrent transactions are executed.

*→ The best schedule is sequential scheduling

A: 1000

A: 950

Buffer 855 A

Buffer 1645 B

temp 95

B: 1500

1645

855

Serializable Schedule ≃ Serial Schedule

22/05/2023

Serializability:

→ Even if the schedule is not sequential but it preserves the consistency of the database.

2 processes

T_i	T_j
I	J
read(Q)	read(Q)
write(Q)	read(Q)
read(Q)	write(Q)
write(Q)	write(Q)

Conflict with eachothers.

* Swapping of instructions is done only when there is conflict b/w the instructions

* Precedence graph: a direct graph where the vertices are the transaction names.

- Set of transactions: $T_1, T_2, T_3, \ldots T_n$.

* If there is an arc b/w then the 2 transaction conflict with each other.

* If there is a cycle in the precedence graph of transactions; then such transaction cannot form ~~serializable~~ conflict serializability

* Recoverable Schedules:

If a transaction T_i reads a data item previously written by a transaction T_j, then the commit operation of T_i must appear before the commit operation of T_j

* Cascading Rollbacks:

A single transaction failure leads to a series of transaction rollbacks.

T_{10}	T_{11}	T_{12}
read(A)		
read(B)		
write(A)	read(A)	
	write()	
		read(A)
abort		

→ If T_{10} fails, T_{11} & T_{12} must also be rolled back

* If a given schedule is not recoverable then such schedule cannot have chance of a cascading rollback

25/05/2023 *CURSORS:

→ Cursor is a pointer to the work area.

→ Oracle creates a memory area, known as context area for processing an SQL statement.

→ A cursor is a pointer to this context area. PL/SQL controls the context area through cursor.

→ 2 types
- → Implicit
- → Explicit

* Implicit cursors are automatically created by Oracle whenever an SQL statement is executed when there is no explicit cursor for the statement. Programmers cannot control implicit cursors.

*→ Explicit cursors are programmer defined cursors for gaining more control over the context area.

An explicit cursor should be defined in declaration section of PL/SQL block.

It is created on a select statement which returns more than one row.

→ SYNTAX:

CURSOR cursor_name IS select_statement;

*Working with explicit cursor:

→ Declaring the cursor.

→ Opening the cursor for allocating memory

→ Fetching the cursor for retrieving data.

→ Closing the cursor to release allocated memory.

Example:

* CURSOR c_customers IS SELECT id, name, address from customers; → declaring

* OPEN c_customers; → opening

* FETCH c_customers INTO c_id, c_name, c_addr; → fetching

* CLOSE c_customers;

We need to use diff. select statements for different cursors.

*Attributes.

%FOUND →

%NOTFOUND →

%ISOPEN →

```
* declare
  cursor c1 is select * from dept;
  cursor c2 is select * from emp;
   s.emp.sal %type;
   begin
   for i in c1
   loop
   s:=0;
  dbms_output.put_line('----------');
  dbms_output.put_line('Department is:'
                        ||i.deptno ||'Department
                         name is: '||
                         i.dname);
  dbms_output.put_line('----------');
  for j in c2
   loop
   if (i.deptno=j.deptno) then
   s:=s+j.sal;
   dbms_output.put_line().empno|| " || j.ename||
                        " || j.sal);

   end if;
  end loop;
  dbms_output.put_line('----------');
  dbms_output.put_line('Total salary is: '||s);
  dbms_output.put_line('----------');
  end loop;
  end;
```

* PL/SQL block using cursor to find employee with given job and deptno:

```
* declare
  cursor c1(j varchar2, dn number) is select
   empno, ename from emp where job=j and
   deptno dn;
   row1 emp%rowtype;
   jb emp.job%type;
   d emp.deptno%type;

   begin
   jb:='&jb';
   d:=&d;
   open c1 (jb,d);
   fetch c1 into row1.empno, row1.ename;
   if c1%not found then
   dbms_output.put_line('Employee does not
                         exist');

   else
   dbms_output.put_line('empno is: '||row1.
                         empno||" "||'employee
```

```
name is:'|| row1.ename);
end if;
end;
```

* Write the cursor PL/SQL program to find manager names in each department.

* Write the cursor program to fetch employees and count no. of employees in each department.

* Write a cursor to display the list of employees who are working as a manager or analyst.

```
declare
cursor c(jb, varchar2) is select ename from
                emp where job=jb;
em emp.job%type;
begin
open c('MANAGER');
dbms_output.put_line('Employes working as
                managers are:');
loop
fetch c into em;
exit when c% not found;
dbms_output.put_line(em);
end loop;
close c;
open c('ANALYST');
dbms_output.put_line('Employees working as
                analyst are:');

loop
fetch c into em;
exit when c% not found;
dbms_output.put_line(em);
end loop;
close c;
end;
```

26/05/2023

→ ~~Cascadeless Schedules.~~

→ Starvation occurs if the operation is waiting for the resources for a longer time; since the operations, which are compatible takes the resources as and when they occur.

*→ Locking protocols is a set of rules followed by all transactions while requesting & releasing locks.

A = 1000 ; B = 500 A+B = 1500

Ex

T_1	T_2
lock-X(A)	
A = A-50	
write (A)	
unlock(A)	
	lock - S(B)
	Read (B)
	~~lock-X(A)~~
	unlock(B)
	lock - S(A)
	Read (A)
	unlock (A)
	display(A+B)

Inconsistent here due to this

lock - X(B)
B = B + 50
write (B)
unlock(A)
unlock(B)
commit

A+B = 1500

commit

A+B = 1450 → Inconsistency

* 2 - PHASE LOCKING PROTOCOL:

→ Growing phase
- → Transaction may obtain locks.
- → Transaction may not release locks

→ Shrinking phase
- → Transaction may release locks
- → Transaction may not obtain locks.

→ Lock can be changed from Exclusive to shared locks, but no new locks are acquired by the data sets.

→ Aging is one of the sol^n for starvation

→ Early unlocking leads to data inconsistency if we unlock; before the commit statement

which may lead to deadlock.

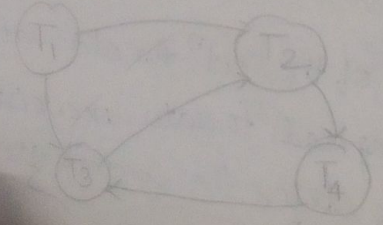
read (D), write (D) → for acquisition of locks.

→ Deadlock occurs when we unlock the data objects at the end of the given process or when circular wait is developed.

→ If any process is killed or aborted; all the locks on certain data objects are released automatically.

→ We draw wait - for graph; to detect deadlock.
↳ dynamic graph which keeps on changing

$T_i \rightarrow T_j$ → this edge exists only if T_j holds a lock on data resource which is requested by T_i which is incompatible lock with T_i.



→ Deadlock

→ If T_2 is killed; T_3, T_4, T_1 are executed.

* Deadlock Prevention Strategies (based on timestamp).
  - → wait-die → non-preemptive
  - → wound-wait → preemptive.

→ Wait-Die:

Older transaction may wait for younger one to release data item. Younger transactions never wait for older ones they are rolled back.

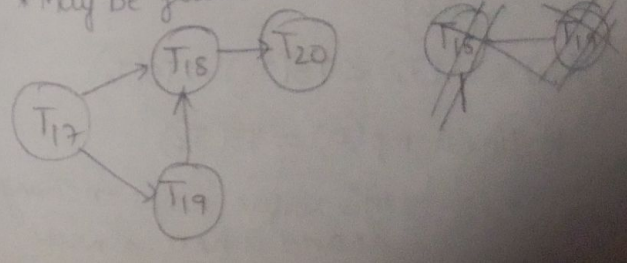
Older transaction → less time stamp & vice versa.

→ Wound-Wait: (force rollback)

older transaction wounds of younger transaction instead of waiting for it.

Younger transactions may wait for older ones.

* May be fewer rollbacks than wait-die scheme.



30/05/2023

* Time Stamp-Based Protocols:

↳ time when the transaction starts.

~~TS(~~ denoted $TS(T_i) < TS(T_j)$

$T_i \rightarrow$ older transaction

$T_j \rightarrow$ new transaction.

*→ The protocol manages concurrent execution such that the time stamps determine the serializability order.

2 types:

→ W-timestamp(Q): largest timestamp of any transaction that executed write(Q) successfully.

→ R-timestamp(Q): largest timestamp of any transaction that executed read(Q) successfully.

Ex: $TS(T_1) < TS(T_2) < TS(T_3) < TS(T_4)$

∴ W-timestamp(Q) = $TS(T_4)$

↳ this value keeps on changing as and when a new transaction updates the value of Q.

i.e. W-TS / R-TS takes the TS(T_{latest}) timestamp of latest transaction).

→ The time-stamp ordering protocol ensures that any conflicting read & write operations are executed in timestamp order.

~~→ TS~~

* If transaction T_i issues a read(Q)

1) $TS(T_i) \leq$ W-Timestamp(Q); then T_i needs to read a value of Q that was already overwritten. Hence the read operation is rejected & T_i is rolled back.

2) $TS(T_i) \geq$ W-Timestamp(Q); then the read operation is executed and R-TimeStamp(Q) is set to max(R-TS(Q), $TS(T_i)$).

* If transaction T_i issues write(Q).

1) If $TS(T_i) <$ R-TS(Q); then the value of Q that T_i is producing was needed previously & the system assumed that

value would never be produced.
Hence write(Q) is rejected

2) $TS(T_i) < W\text{-}TS(Q)$; hence write(Q) is rejected and

3) $TS(T_i) > R\text{-}TS(Q)$ & $TS(T_i) > W\text{-}TS(Q)$:
write(Q) is executed and
$W\text{-}TS(Q) = TS(T_i)$.

→ TimeStamp based protocols leads to conflict serializable schedule.
- → There are no cycles in the precedence graph.
- → Ensures freedom from deadlock

* Lock Based & TimeStamp based protocols ensures the property of isolation.

*→ Log Based Recovery:
- → helps in creating a log write operation and holds the intermediate values.

* Recovery ensures the properties of atomicity & durability

→ RECOVERY.

* Transaction failure
- → Logical errors
- → System errors (deadlock)

* System Crash: power/hardware/software failures leads to system crash.

* Disk failure: a head crash/similar disk failure destroys all or part of storage.

→ RECOVERY: 08/06/2023

* Transaction failure
  - → Logical errors
  - → System errors (deadlock)

* System Crash: power/hardware/software failures leads to system crash.

* Disk failure: a head crash/similar disk failure destroys all or part of storage.

→ Recovery system helps the transaction manager to retrieve back the data which was held before the transaction failure or system failure occurs.

* Storage Structures:

→ Volatile storage ⇒ does not survive sys. crash Ex: Main Memory/Cache Memory

→ Non-volatile storage ⇒ survives sys. crashes Ex: disk, tape, flash memory

→ Stable storage ⇒ a mythical form of storage that survives all failures.

Ex: maintaining multiple copies on distinct media.

* Recovery Algorithms:

1. Recovery data when the Transaction is being executed

② Whenever a transaction failure occurs, we must follow ① and recover data.

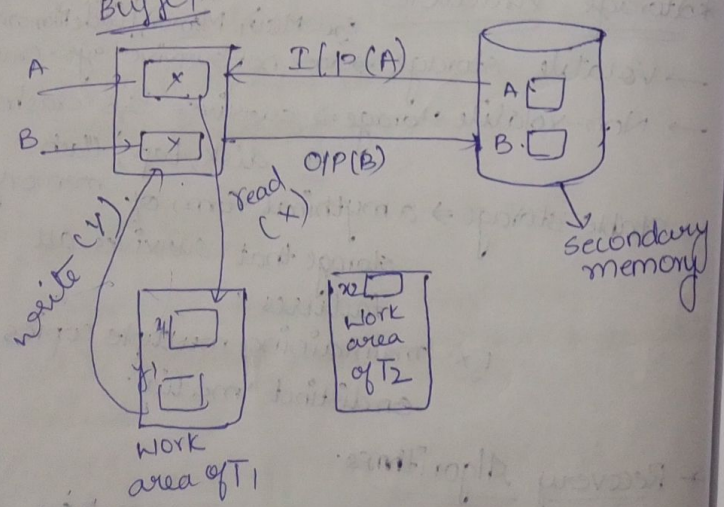
Both ~~these~~ the steps are common for any recovery system.

① → deals with ensuring enough info exists to recover from failures

② → ensures atomicity, consistency & durability.

* Data Access:

Physical blocks → address blocks residing on disks.



Buffer blocks: Blocks residing in main memory

* Log Based Recovery:

→ kept on stable storage.

→ Log is a sequence of log records & maintains a record of update activities.

Ex: $\langle T_i, X, V_1, V_2 \rangle$

$V_1 \rightarrow$ old value
$V_2 \rightarrow$ new value } value of X in transaction T_i

PL/SQL programs
Normalization
Transactions
Concurrency Control
Recovery system

15/06/2023

* INDEXING AND HASHING:

→ A set of data can have a no. of indices; but in those one index is called primary index.

* Types of hashing:

→ Static hashing: fixed no. of buckets.

→ Dynamic hashing: variable no. of buckets (buckets can be expanded or compressed dynamically)

* index file consists of records of the form

Search-key	pointer

↓ points to the next record.

* Types of indices
  - → Ordered indices (sorted order)
  - → Hash indices. (hash function is used).

* Index evaluation metrics:

→ Access types supported efficiently.
→ Access time → Insertion time
→ Deletion time → Space overhead.

→ If ordered index:

→ primary index (Cluster index): Searches for a key sequentially and usually the search key is primary key.

→ secondary index: searches for a key in an order different from sequential order.

→ ~~Hash~~

→ Dense Index. searching is easy & space is more

→ Sparse Index: Contains index records for only some search key values.
↳ Less space.

→ Multi-level index:

* If primary index does not fit in memory, access becomes expensive.

→ Treat primary index kept on disk as a sequential file and construct a sparse index.

* outer index: sparse index of primary index.

* inner index: primary index file.

20/06/2023

* → Open hashing | Dynamic hashing helps in ↑sing or ↓sing the no. of hash buckets according to the requirement.

→ Hash function generates values over a large range – typically b-bit integers with $b = 32$.

* Length of prefix be i bits $0 \le i \le 32$.

→ Bucket address table $= 2^i$

* 2, 3, 5, 7, 11, 17, 19, 23, 31

$h(x) = x \bmod 8$

$h(2) = 2 \rightarrow 010$

$h(3) = 3 \rightarrow 011$

$h(5) = 5 \rightarrow 101$

$h(7) = 7 \rightarrow 111$

$h(11) = 3 \rightarrow 011$

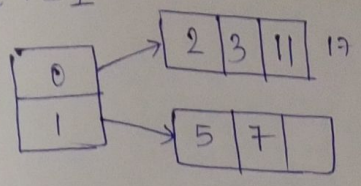
$h(17) = 1 \rightarrow 001$

$h(19) = 3 \rightarrow 011$

$h(23) = 7 \rightarrow 111$

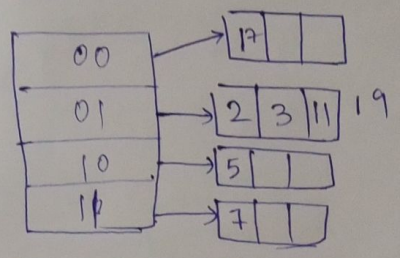
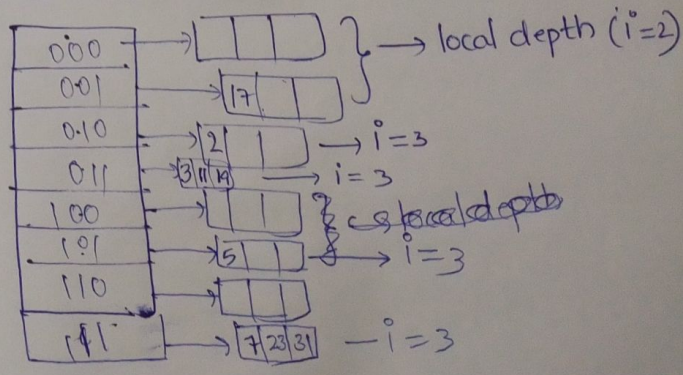
$h(31) = 7 \rightarrow 111$

at $i = 1 \rightarrow 1$ MSB



In the above case; if it is static hashing we add the overflow bucket; but in extendable hashing; we extend the hash table (∵ 17 cannot be stored).

$i = 2 \rightarrow 2$ MSB



at $i = 3 \rightarrow 3$ MSB