CSE 411 / L-21 / 11.11.2024 /

Slide - 30 - 37
Code - Part

Reference Type:

⇒ ref (Person) scope people
(Person → Type, people → Table)

Person (ID), Without joining two table, we can now access any attribute directly.

⇒
```
SELECT  dept_name, head → name
FROM    department
```

Write a SQL to find dept_name, head name and address.

⇒
```
SELECT  dept_name, head -> name, head -> address
FROM    departments
```

Example from slide - 33

(a)
```
create type Customer (
        c-id  varchar(20) primary key,
        name  varchar(30),
        balance  int,
        ref from (c-id));
create table cust of Customer;
```

```
create type Purchase (
        item varchar(20),
        purchased-by ref (Customer) scope cust );

create table purch of Purchase;
```

(b)

```
insert into cust values
        ("C001", 'Rafique", 5000),
        ("C002", "Atique", 10 000);
```

(c)

```
insert into purch values
        ("watch", "C002");
```

```
create table people of Person
ref is person_id system generated;
```

} System automatically generate the unique ref for each tuples.

- good for large dataset.

Example - from slide-36

```
insert into departments values ('EE', null)
```

```
update departments
    set head = ( SELECT ref(p)
                 FROM people as p
                 WHERE name = "Fatema")

    WHERE dept_name = 'CE'
```

Object-Relational Mapping (ORM):

- maps object to relations
- system we maps to fetch relevant data from relations and construct object.
- after update, objects are stored back to the database.

Hibernate ORM:

- Provides API
- queries translated to SQL.
- overheads for bulk update.

Comparison of O-O and O-R Database | Slide-39

L-22 / 13.11.2024 /

No SQL (Big data)

- Big data is used for data analytics and on-line querying.

- A NoSQL database provides a mechanism for storage and retrieval of data that use looser consistency models than traditional relational databases in order to achieve horizontal scaling and higher availability.

  increase server one by one in parallel.

  - its also allow SQL language to be used.

- GFS ⇒ Google File System

  - No schema
    - use index
  - change of developer will cause difficulty to modify the database structure.

- NoSQL Components:

  - Analytics Interface : Pig, Hive
  - Imperative Language : RoR, Java, Scala
  - Data Parallel Processing : MapReduce, Hadoop
  - Distributed key, value, or column store : Cassandra, Hbase, Redis.
  - Scalable File System : GFS, HDFS

NoSQL Features:

- Scalability is crucial ⇒ rapid increase
- Large servers are expensive

} solutions:
- use clusters of small commodity.
⇒ Cloud-based storage
- use of replication, sharding.

- Not a well defined schema
- allow for semi structured data.

Slide-8

Flavors of NoSQL:

⇒ 4 main type:

(i) key value stores
(ii) document database
(iii) column family (big-table) stores
(iv) graph database.

MongoDB:

- semi-structured data model
  - XML
  - JSON or BSON
- support sharding (→ scalability) and replication. (→ availability)
- Relation Table ⇒ collection
- Table Row ⇒ document
  ↳ each document can have a different set of attribute, even in the same collection.

Example - Slide-15

- document based queries.
- can create an index on any attribute.

Slide - 17, 18

* _id field:
  ⇒ must have in every document
  ⇒ must be unique within the collection
  ⇒ act as primary key of the collection.

* Relationship in MongoDB:
  - store reference to othe documents using their _id values.
  - embed documents within other documents.

Slide - 22

* Differences:

⇒ NoSQL:
  (i) tables are unnatural
  (ii) joins are evil
  (iii) need to be able to 'grep' my data

⇒ DB:
  (i) Tables are natural
  (ii) data independence ⇒ we can pre-compute joins under the covers
  (iii) this is the price of all the DBMS goodness we get.

L-23/18.11.2024/

Data Warehouse Design

* Data analytics: ⇒ BI: Business intelligence.
  - used to make business decisions.
    ⇒ DSS: Decision Support System ⇒ focus on reporting and aggregation
  - what product to suggest for purchase
  - what product to manufacture

- Steps:
  (i) extract ⇒ from multiple source
  (ii) transform ⇒ to common schema
  (iii) load ⇒ into data warehouse
  } ETL or ELT

* OLAP: Online analytical processing system
  - allow interactive querying

* R/SAS/SPSS:
  - used for statistical analysis.
  - parallel processing of big data

⇒⇒ Build predictive models ⇒ Machine Learning, Data mining
  - used for decision making
    - predict to make loan decision
    - predict how much to stock
    - from past history of sales, predict future sales

Federated database system:

- software layer
- designed to manipulate information in heterogeneous database.

Slide-9 Diagram

Schema integration:

- Global schema integrates all the local schema
- queries issued against global schema will translated to queries on local schema.

⇒ Wrapper:
- translate data from local to a global schema.
- must translate updates on global schema to updates on local schema.

Slide-11 Diagram

⇒ Mediator:
- support common schema and queries
- not updates.

Data Warehouse:

- alternative to data integration

Diagram - Slide-13

- migrate data to a common schema
- avoid run time overhead
- cost of translating schema to a common warehouse, can be significant.

- like a repository or archive of information
- data are stored for a long time.
  - as a historical data

What schema to use?

⇒ data collected from different schemas may have different data models. we need to convert the data to the integrated schema before they are stored.

Data cleansing:

- correct and preprocess the data
- correct numerous minor inconsistencies.

Data transformation:

- transform from host format to warehouse format.

Multidimensional Data & Warehouse Schemas:

- use OLAP tools
- relations are classified as fact tables and dimension tables.
  - fact tables record information about individual events and are usually o very large
  - fact table will increase in tupple counts
  - other dimension tables data are fixed.

- attributes in fact table are classified as dimension attributes or measure attributes.

stone quantitative information

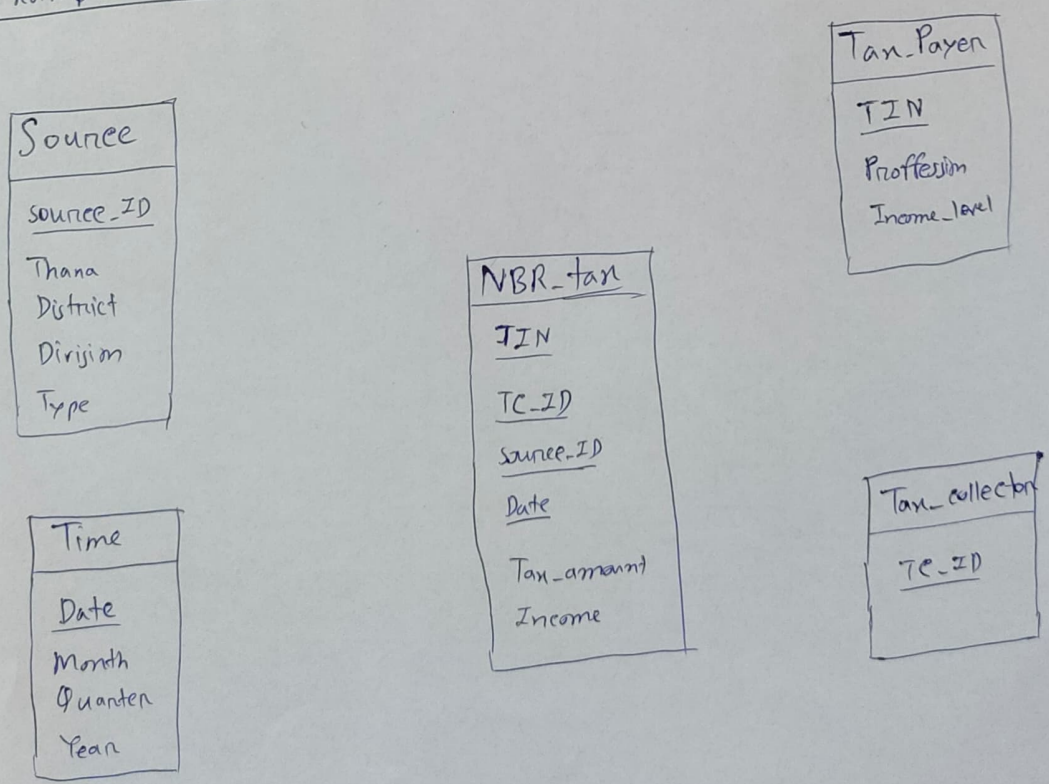
* Data that can be modeled using dimension attributes and measure attributes are called multidimensional data.

L-24 / 20.11.2024 /

Quiz-2

* Designing a Star Schema:

Example - Slide-20



Slide-21, 22

OLAP: Online Analytical Processing (~~OLAP~~)

- interactive analysis of data.

sales(item_name, color, clothes_size, quantity)

⇒ how can we get this table?

SQL:

```
SELECT  a.itemname as item_name, a.color , a.size as clothes_size,
                                                b.number as quantity
FROM  item_info a, sales b
WHERE  a.item_id = b.item_id
```

Cross-tabulation or Pivot table: Data Cube (Slide-27)

color

item_name	dark	pastel	white	total
skirt	8	35	10	53
dress	20	10	5	35
shirt	14	7	28	49
pants	20	2	5	27
total	62	54	48	164

```
SELECT item_name, sum(quantity)
FROM sales
GROUP BY item_name
```

```
SELECT sum(quantity)
FROM sales
```

```
SELECT item_name, color, sum(quantity)
FROM sales
GROUP BY item_name, color
```

```
SELECT color, sum(quantity)
FROM sales
GROUP BY color.
```

L-25/25.11.2024

Hierarchies on Dimensions:

- lets dimensions be viewed at different levels of detail.

Slide-28

Find Report on Month, Quarter, Year:

⇒

R_1 = SELECT Date, Month, Quarter, SUM(quantity)

FROM date_info as d, sales as s

WHERE d.date = s.date

⇒ What is the number of tuples in R1?

⇒ 18

GROUP BY d.date

⇒ Now?

R_1 = 1825

- Sales = 18 → 5 years.
- date = 5 × 365
  = 1825

- R_2: Find Report for Year on total sale ⇒ 5 tuples
- R_3: Find Report for Month, Year wise total sale ⇒ 5 × 12 = 60 tuples
- R_4: Find Report for Day, Month, Year wise total sale

⇒ 7 × 12 × 5

= 420 tuples

R_5 : Find Report for Date, Month, Year wise total sale ⇒ 1825 tuples.

⇒ R_5 :

```
SELECT Date, Month, Year, SUM(sale) as d.total_sale
FROM R1
GROUP BY Date, Month, Year
```

R3 :

```
SELECT Month, Year, SUM(d_total_sale) as month_total_sale
FROM R5
GROUP BY Month, Year
```

R2 :

```
SELECT Year, SUM(month_total_sale) as year_total_sale
FROM R3
GROUP BY Year
```

Find total sale in 2020 :

⇒ use R2 ⇒ only 5 tuples need to process then will show the output.

⇒

```
SELECT total_sale
FROM R2
WHERE Year = 2020
```

L-26/27.11.2024

* customer(CID, name, street, thana, district, division, purchase_amount)

⇒ Total purchase group

Rollup

street, thana, district, division, sum(purchase_amount)

↓

thana, district, division, sum(purchase_amount)

↓

district, division, sum(purchase_amount)

↓

division, sum(purchase_amount)

↓

total, sum(purchase_amount)

* Rollup:

- extension of the GROUP BY clause
- generate multiple grouping sets.

⇒ table(C_1, C_2, C_3)

C_1, C_2: dimension attribute

C_3: measure attribute

SQL: SELECT C_1, C_2, AVG(C_3)
FROM table
GROUP BY ROLLUP(C_1, C_2) → (C_1, C_2), (C_1), ()

table

c_1	c_2	c_3
Dhaka	Sadar	200
Dhaka	Mirpur	300
Dhaka	Bashundhra	500
Khulna	Sadar	100
Khulna	Shibgani	200
...	...	...

After ROLLUP(c_1, c_2):

	c_1	c_2	c_3
(c_1, c_2)	Dhaka	Sadar	300
	Dhaka	Mirpur	500
	Dhaka	Bashundhara	400
	Khulna	Sadar	300
	Khulna	Shibgani	200
	...	...	...
(c_1)	Dhaka	null	400
	Khulna	null	250
	...	...	...
( )	null	null	325

CUBE:

- another extension of the GROUP BY clause.
- generate sub-totals like the ROLLUP

⇒

ROLLUP(c_1, c_2)

(c_1, c_2)
(c_1)
( )

CUBE(c_1, c_2)

(c_1, c_2)
(c_1)
(c_2)
( )

} 2^n

* fact (item_id, day, month, year, sale)

SQL:

```
SELECT day, month, year, SUM(sale)
FROM fact
GROUP BY CUB (day, month, year)
```

⇒ $2^3 = 8$ ⇒ total 8 combination:

day	month	year
day	month	null
day	null	year
null	month	year
day	null	null
null	month	null
null	null	year
null	null	null

* PIVOT Table:

⇒

```
SELECT *
FROM sales
WHERE date between '01/01/2024' and '03/01/2024'
PIVOT ( sum(sale_amount) FOR product IN ('Product A', 'Product B'))
```

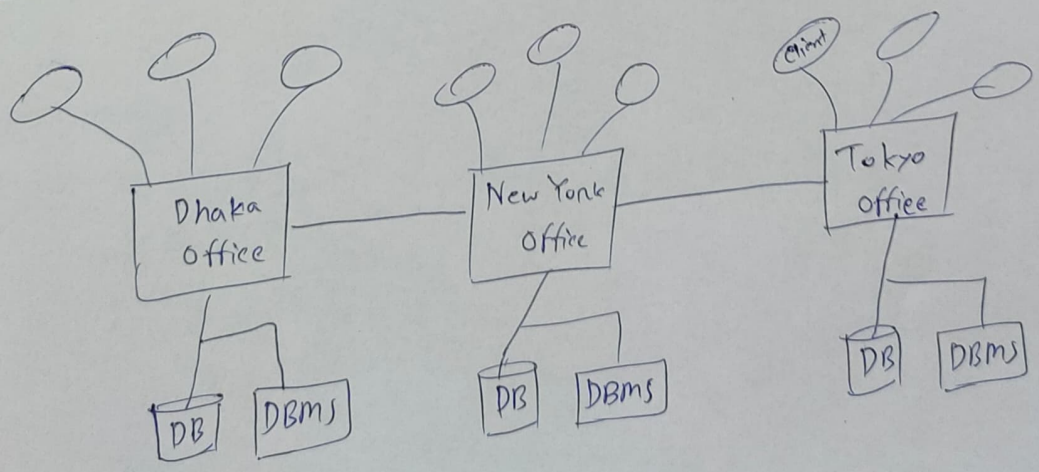
	Product A	Product B	total
01/01/2024			
02/01/2024			
03/01/2024			
total			

Final Exam upto this 17.12.2024

[-27/02.12.2024/

Distributed DBMS

- Multinational Company:

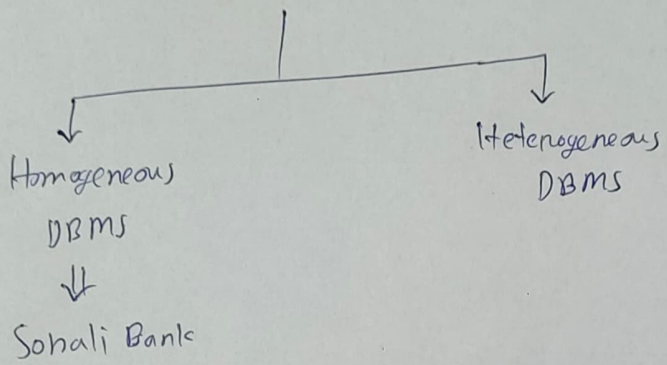


* Reliability ⇒ Replication and Fragmentation

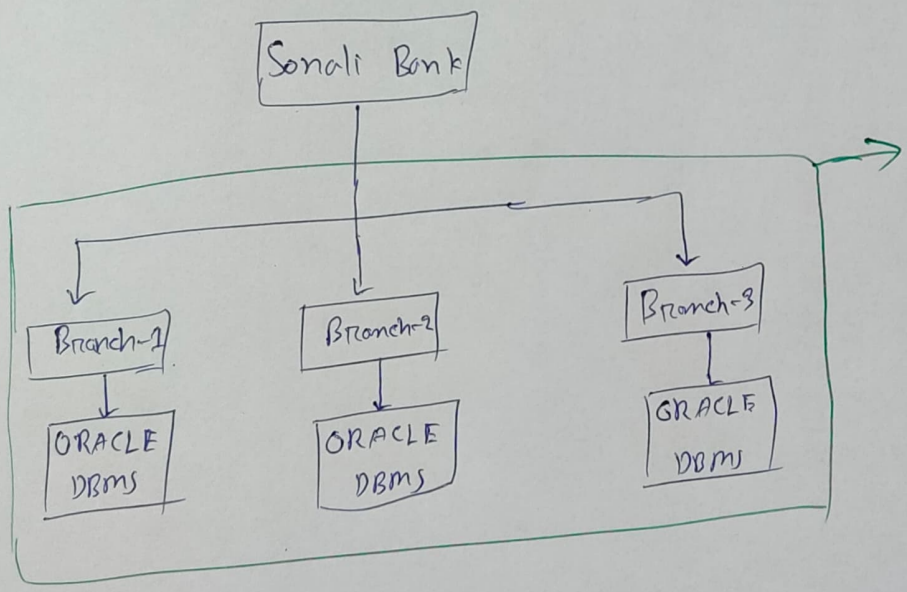
Availability

Dhaka ⇒ Replicated in Tokyo

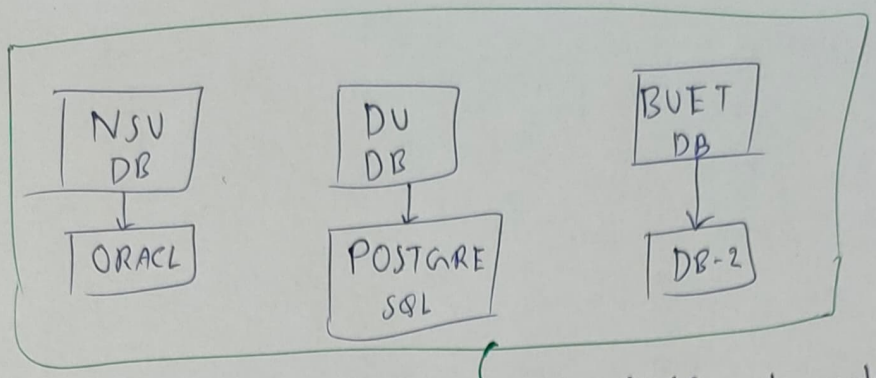
* Distributed DBMS



*

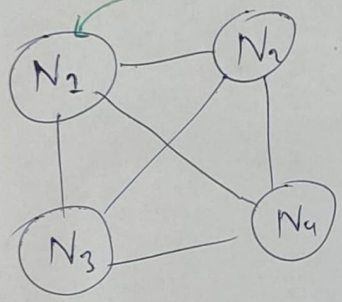


Distributed, same schema used, same data structured.
So, it is homogeneous.



different schema
different data structured
making them distributed will be heterogeneous

Replication:



Person (NID, name, street, city, thana, district, age)

Q1:

```
SELECT *
FROM Person at Side N1
```

⇒ - No transition cost
- Query cost only

Q2:

```
SELECT *
FROM Person at Side N2
```

⇒ Query cost + transition cost.

Fragmentation:

two types:

(i) Horizontal
(ii) Vertical

✳ Person (....)

⇒ Horizontal Fragment on Age

$$Person\_over\_18 \leftarrow \sigma_{age \geq 18}(Person)$$

$$Person\_under\_18 \leftarrow \sigma_{age < 18}(Person)$$

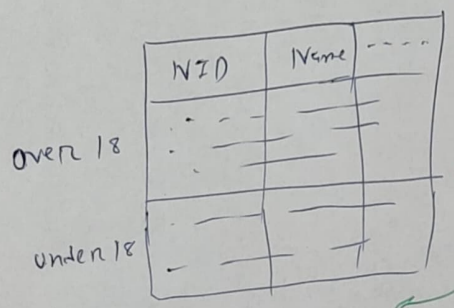
$$Person = Person\_over\_18 \cup Person\_under\_18$$

✳ Find Person age over 30

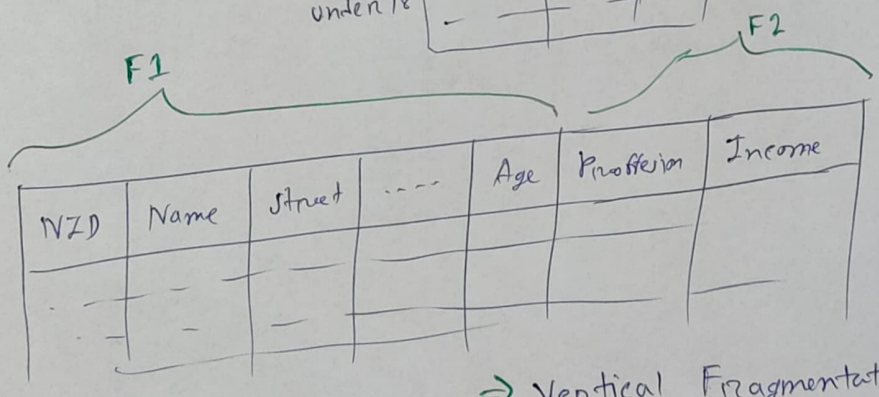
⇒

```
SELECT NID, name
FROM Person
WHERE Age >30
```

```
SELECT NID, Name
FROM Person_over_18
WHERE age > 30
```



⇒ Horizontal Fragmentation



⇒ Vertical Fragmentation

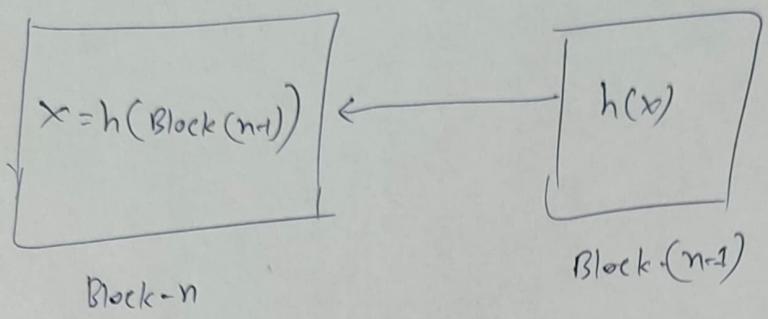
$$Person\_personal\_info = \Pi_{NID, name, street, \ldots, age}(Person)$$

$$Person\_income = \Pi_{NID, profession, income}(Person)$$

↳ must need to be store in all vertical frag... without primary key table will be useless.

L- 28/03.12.2024

Blockchain



(i) $h(x) = h(x')$

(ii) $h^{-1}(h(x)) = x$

} Not possible

L- 29/03.12.2024

Blockchain Overview

Recap a question

Student

Id	Name	Mobile	Member-1	Member-2	Member-3	
1001	Abid, Mahmud, Hague	017... 018...	-	-	-	→ (*) How to handle these null value in relational and NoSQL.
1002	...	...	Com	IEEE	-	
1003	...	...	-	-	ACM	

Student Relational Model

Student

ID	FirstName	MiddleName	LastName	...
1001	Abid	Mohammed	Haque	...
...	...	...	...	...

Student_mobile

ID	Mobile
1001	017...
1001	018...
...	...