DBMS - Short notes

classmate
Date
Page

Unit 1

DBMS provide ability to define, store and manipulate data.

1) Self-describing, 2) Independence of programs and data
3) Data Abstraction, 4) Support multiple views of data
5) Sharing of data and multi-user transaction processing.

There are 2 main categories of users of database

1) Actors on the scene. => Database Adminstrators, Database Designers, End users
System Analysts and Application Developers
2) Workers behind the scene => System Designers, Tool Developers, Operators and
Maintenance Personnel.

Advantages :- (i) Control Redundancy, (ii) Restrict unauthorized access
(iii) Persistent Storage, (iv) Efficient query processing
(v) Backup and Recovery services, (vi) Multiple views to multiple users

Disadvantages :- (i) Need fixed standards, (ii) Reduced Application development time
(iii) Need to maintain flexibility to change data structures.
(iv) Economies of scale (v) Current info must be available immediately

We shouldn't use database when
(i) High initial investment, (ii) DB, Application won't change, (iii) No multiple users
(iv) When real-time deadlines and DB overhead not affordable (v) Complex Data

Data Models = set of concepts to describe the structure of database, operations
for manipulating these structures and certain constraints

Conceptual = Way user perceives data
Physical = How data is stored in computer.
Implementation / Representational = In between both above. Mainly used

Three Schema Architecture

Internal = describe physical storage	Physical Data Model
↓ Conceptual = Structure and constraints for all users	Conceptual / Implementation
↓ External = various user views.	Conceptual Data Model

Data Independence = when lower level schema changes only mappings to upper change

(i) Logical DI = When conceptual schema changes, external need not
(ii) Physical DI = When internal schema changes, conceptual need not.

DBMS Interfaces = Programming Language or User friendly.

Database System Utilities = Load data, back up data, monitor performance
sorting, user monitoring, data compression

Database System Environments

1) Centralized = Everything into 1. DBMS hardware, software, UI, Application Programs.

2) Client-Server = Client has proper UI, API to send to server via network
Server respond to client query.
Both must have proper client and server modules installed.

3) Three-tier = Extra layer called application layer added. as intermediate.
Stores web connectivity software and buisness logic
Client input → Partially Process → Server
Enhances security as client need this to access server and vice-versa

Entities-Relationship Model

Entity = Specific things or Objects in mini-world. ▭ -Strong ▭ -Weak

has attributes ⬭ = Simple, Composite, Multi-Valued ⬭, Derived ⬭

one of which is key ⬭

Entity set = collection of all entities

Value set = Domain of attributes

Relationship relates two or more distinct entities with specific meaning.

Relationship type = schema description (name, entities, type) → 1:n, 1:1, m:n, recursive

Relationship set = set of relationship instances

Cardinality ratio = ratio of maximum participation of participating entities

Alternative of this is (min, max) notation

Total participation = When (≥1, ≥1) for the relation ═

Partial participation = When (0, ≥0) ─

Weak entity = no key and identification depends exactly on another.
▭, Partial key ⬭

DBMS Short notes

Unit 2

Relational Model :- This is what RDBMS follow.

Relation = Table Rows = Tuples Column = Attributes

Key = Attribute that uniquely identifies each row in table = Primary Key

Schema = $\{R_1, R_2, \ldots, R_n\}$ => set of relation

Relation = $\{A_1, A_2, \ldots, A_m\}$ => set of attributes

Domain = data type or format that attribute has to follow

Relation State = subset of Cartesian Product of domains of attributes. It is the populated table.

Relational Model was needed to remove all negatives of flat file. Flat file had a lot of redundant data and modifying data was tough. We split the single table db into relational db using normalisation

Constraints of Relational Model

(1) Inherent or Implict :- Based on data model itself: Eg. No lists as values

(2) Schema Based or Explicit :- Expressed using schema, Provided by model.

(i) Domain Constraint :- Attribute value must be in domain specified

(ii) Key Constraint :- Key value must be unique for every row

(iii) Entity Integrity :- Key value can never be NULL

(iv) Referential Integrity :- Foreign Key must point to valid primary key.

(3) Semantic :- Cannot be expressed by model.

Insert, Update, Delete Violations

Insert can violate all 4 Schema constraints,

Delete can violate only referential. => It must be done with a certain extra input of whether to RESTRICT, CASCADE or SET NULL to ensure how delete is handled if referential integrity is broken.

Update = Insert + Delete.

Mapping of ER diagram to Relational

1) Mapping entities and weak entities to relations. Weak entities will have a foreign key in them pointing back to their strong entity primary key

2) 1 : 1 relations are found. Assuming S has total participations and T isn't. Primary key of T put into S as foreign. Can also merge S, T or create new R.

3) 1 : n relations are found. S = 1 T = n. Put primary key (S) into T

4) m : n relations are found. Create new relation R having primary key of both

5) Multi-valued attributes will also get a new relation with key same as primary key

6) N-ary relationship we create a new relation containing all primary keys.

Relational Algebra

Basic set of operations for relational model. It is closed. Result of operation is always a new relation.

Unary Relational Operators

(1) Select $\sigma_{<conditions>}(R)$

This returns R' that has all the rows that match boolean condition specified.

(2) Project $\pi_{attribute\ list}(R)$

This returns R' with all rows but only specified attributes.

(3) Rename $\rho_{New\_name(new\_attributes\ names)}(R)$

This returns R' itself with new name and attributes.

Set theory operators

(1) Union	$R \cup T$	All tuples either in R or T	$n_r + n_s$ max
(2) Intersection	$R \cap T$	All tuples in both R, T	n_r or n_s max
(3) Set Difference	$R - T$	All tuples in R but not in T	n_r max
(4) Cartesian Product	$R \times T$	All possible combinations of R and T combining.	$n_r * n_s$ elements

Pre-requisite for Union, Intersection, Difference is that both R and T have to be type compatible ⇒ same no. of attributes, corresponding pair have same domain. Their names in result will be same as R1

Binary Relational Operations

(1) Join ⋈ $R \bowtie_{<join\ condition>} T$ (Theta join)

Will return tuples that match the join condition.

$R.A_i > T.B_j$ or $<$, or $=$ if $=$ it means equi-join

(2) Natural Join * $R * T$

Will perform equi-join on the basis of matching column names.

It will remove repeated superfluous attributes.

(3) Outer Join ⟗ $R \text{ ⟗ } T$

This will keep all the tuples. Matched + R with S as null + T with R as null.

Left ⟕ will keep only R Right ⟖ will keep only T

(4) Outer Union

This for union when not type compatible. All additional kept

(5) Division $R \div T$ $R(A, B, \ldots) \div T(A)$

This will return a tuple from R if it has all occurences of A in T as part of R.

Query Tree Notation

Internal data structure to represent query. Estimate work involved, Optimizing it
Nodes = Operations Leaf nodes = Relations.

Additional Relational Operators

(1) Aggregate functions $\mathcal{F}_{<aggregate\ fn>\ attri}\ldots\ (R)$
Sum, Count, Maximum, Minimum, Average etc.

(2) Grouping with Aggregation grouping attr $\mathcal{F}_{<aggregate>\ attr}$
This will group results based on an attribute given. We need not apply any aggregate function and just group alone as well.

Not defined by relational algebra

(1) Recursive closure.
Recursive relationships that require iteration to find a hierarchy or something similar

classmate
Date
Page

Unit 3

We have to learn basic SQL commands here.

- **Create Table** table_name
  (Vari DT Constraints,
  ⋮
  Primary Key (Vari i)
  Foreign Key (Var j) references TableName (Var)
)
  DTs = Numeric, Char, Bit, Bookan, Date, Timestamp, Interval [Increment/Decrement time]
  Binary LOBs or Character LOBs
  *Internal LOBs also provide transactional support.
  Constraints = Primary Key, Foreign Key [Specified separately]
  Default, NOT NULL, Check (Cond) [Specified with attribute]
  We can name constraints and specify at end while creating table
  Constraint cons_name condition
- **Drop** command. This is used to drop named schema elements. We have to provide options of either cascade or restrict.
- **Alter** command. This can change named schema elements
  Alter Table Table_Name ADD/Drop Column/Constraint
- **Insert** command. This is used to add values into tables.
  (i) Insert into Table_name values (- - -)
  (ii) Insert into Table_name (- - -) QUERY
  (iii) Create Table Table_name LIKE Table_1 (QUERY) WITH DATA
- **Delete** command. This is used to delete certain tuples
  (i) Delete From Table_name Where Condition.
- **Update** command. This is used to modify attribute values of selected tuple
  (i) Update Table_name Set Var= Where Condition
- **Three-Valued Logic**
  The two values used here are True, False, Unknown/Null

AND	T	F	N
T	T	F	(N)
F	F	F	F
N	(N)	F	N

OR	T	F	N
T	T	T	T
T	T	F	(N)
T	F	(N)	N

NOT	
T	F
F	T
N	N

SQL Retrieval Queries

Select <attribute> * = all Distinct = Unique values for attribute Aggregate variables

From <tables> as Alias/Another Name

[Where <conditions>

[Nested/Sub or Correlated Query]]

[Group By <grouping attributes>] Partition result into subsets. must be in select NULL is also a group

[Having <group condition>] - condition on group by

[Order By <attribute list> DESC/ASC]

(i) If Where is not specified then Cartesian Product

(ii) Comparison Operators = Mostly usable in only Where clause

a) Like = Pattern matching for strings 'Regex'

b) Between __ AND __ for variables

c) In used to check if value In set of values

d) Any/Some when we want other conditions other than equal. >, >=, <=, <

e) All (Query) when we want to compare with all rather then 1

(iii) Nested Query => Complete Query within another Query.

This could be uncorrelated or correlated. If uncorrelated it runs only once throughout execution of outer query.

Correlated Nested Query => Whenever a condition in the where clause of a nested query, it references attribute of a relation declared in outer query.

The correlated query is executed once for every tuple in outer query.

(iv) Functions in Where clause

a) Exist (Query) True = if 1 record at least False = if no records

b) Unique (Query) True = if all records unique False = if not all unique

(v) JOIN conditions in From Clause

We can say

From Table1 AnyType Of Join Table2 ON Condition

*Natural join has no condition necessary.

(vi) Constructs

a) With name_for_temp_table As (Query / Insert / Update / Delete)

These create an auxillary table for querying in larger queries.

They form Common Table Expressions.

The name_for_temp_table can be same as existing table and will be given higher precedence.

b) Case

When Condition Then Action

⋮

Else Action ;

- **Assertions** = Used to specify additional types of constraints which is outside scope of built-in relational model constraints. This specifies a query that selects any tuples that violate desired condition.

Create Assertion Name Check (Condition using Query)

Condition will always be enforced and any violation raises an error

- **Triggers** = Specify automatic actions that will be performed when certain events or conditions occur.

Create Trigger T_name

Before/After Insert/Update/Delete Of Attributes On Table

For Each Row

When Condition

Action

Event ⟶ Condition ⟶ Action

This gives use an object called New which can be used to check changes.

- **Views** = Single Table derived from other tables! It does not physically exist

Create View View_Name (Attributes) As Query.

We can implement either query modification [Change query on view to tables] which is very inefficient or view materialization where we create a temporary view table when it is first queried. We would need a good strategy to update this using immediate, lazy or periodic updates.

These can be used as an authorization mechanism.

- **Functions**

Create Function fn_name (param_list)

returns return_type — Must have return type

language plpgsql — Can't perform transactions.

as — Don't have transactional support.

$$ declare --variables $$

$$ begin --logic ; end ; $$

- **Stored Procedures**

Create Procedure procedure_name(param_list) — No compulsion of return type
language plpgsql — Can make use of transactional
as — support
$$ declare --variable declaration $$
$$ begin --logic end; $$

- **Cursors** = Allows us to encapsulate a query and process each individual row at a time. To divide large result set to parts.

Declare → Open → Fetch row by row → Process → Close

These are mainly used in functions and stored procedures.

Declare	Declare cursor_name refcursor; OR cursor_name cursor[attributes] for query;
Open	Open cursor_name for query; OR Open cursor_name [(name:=value, ...)]
Using	Fetch [direction {from | in}] cursor_name into target_variable next, last, prior, first, Absolute index, Relative count.
	Move [direction {from | in}] cursor_name
Modifying DB	Update table_name Set column = value Where current of cursor_name;
	Delete from table_name Where current of cursor_name;
Closing	Close cursor_name;

- **Online Analytical Processing**

classmate
Date
Page

Unit 4

Design Guidelines for RDs

1) Each tuple should represent one entity. Attributes of different entities should not be mixed together in same relation.
2) Design a schema with no insert, delete, update anomalies.
   (a) Update anomaly = Data inconsistency from data redundancy and partial update
   (b) Insert anomaly = Inability to insert due to absence of data.
   (c) Delete anomaly = Unintended loss of data during delete.
3) Tuples should have minimal null values
4) Relations should satisfy lossless join condition.
   ↓ no spurious tuples should be generated by natural join of all.

Functional dependencies = specify formal measures of goodness of relational designs. These are constraints that are derived from the meaning and inter-relationships of data attributes. $X \to Y$.

We may conclude that a FD may exist but not confirm. However, we can effectively say that certain FDs don't exist at all.

For $X \to Y$ if $t1[X] = t2[X]$ then $t1[Y] = t2[Y]$ is a must.

Given a set of FDs F, we can infer additional FDs that hold.

Armstrong Rules (Sound and Complete, Can derive all others)
1) Reflexive = If Y is subset of X, $X \to Y$ always holds
2) Augmentation = If $X \to Y$ then $XZ \to YZ$
3) Transitive = If $X \to Y$, $Y \to Z$, $X \to Z$ holds.

Decomposition $X \to YZ = X \to Y, X \to Z$ Union $X \to Y, X \to Z = X \to YZ$

Pseudotransivity $X \to Y$, $WY \to Z$ then $WX \to Z$

$X \to Y = XW \to YW$ then $XW \to YW$ $YW \to Z = XW \to Z$

Closure of a set F of FD's is set F^+ of all FDs that can be inferred by F
of a set X of attr on FD's F is X^+ of all attributes derived from X.

Equivalence of set of FDs = If every FD in F can be inferred from G and every FD in G can be inferred from, then $F^+ = G^+$ F covers G, G covers F

Minimal covers of FDs = If every FD, RHS = 1 attribute [cannonical form]
Can't replace $X \to A$ with $Y \to A$ where Y is subset of X
Can't remove any FD and have equivalent to F.

Algorithm) 1st) Get it to cannonical form
2nd) For each $X \to A$, B in X, if $F - \{X \to A\} \cup (X - B) \to A = F$ remove B
3rd) For each $X \to A$, if $F - \{X \to A\} = F$ remove $X \to A$.

Key and Candidate Keys

Key = attribute/set of that uniquely identifies every row = Known

Candidate Key = Minimal Super Key.

To find candidate keys

1st) Determine all essential = Don't appear at RHS at all

2nd) If all essential derive non-essential ∴ 1 key skip 3 else goto 3

No. of super keys $\underbrace{\_\_\_\_}_{\text{essential}}$ $\underbrace{\_\_\_\_}_{n = \text{non-essential}}$ 2^n

3rd) Have to check possible combinations

Normalisation = The process of decomposing unsatisfactory relations by breaking up their attributes into smaller forms.

1st Normal Form = We disallow composite, multivalued, nested relations

↓ remove it ↓ make more records ↓ decompose it to 2.

2nd Normal Form = Uses concept of full functional dependency

Full Function Dependency => $Y \to Z$ and removing anything from Y means $Y \not\to Z$

If every non prime attribute A in R is fully functionally dependent on primary key

Non-prime => not in any candidate key

This usually fails when primary key is composite key. ✓ if key = 1 attribute

3rd Normal Form = Uses concept of transitive functional dependency.

If $X \to Y$, $Y \to Z$ with X as primary and Y is not candidate key, we need to split. If Y is candidate key ✓ no problem.

For all FD $X \to A$, if X is superkey or A is prime attribute.

If every non-prime attribute is fully functional dependent on R and not transitively dependent on R.

Boyce-Codd Normal Form = $X \to A$ means X must be superkey. We remove condition of it being candidate key = ✓ to X

* A relation R(A, B) is always in BCNF.

BCNF will look to preserve lossless join but is willing to give up preservation of functional dependency.

Given R not in BCNF and $X \to A$ be the FD that violates. Then decompose as $(R - A)$, $(X \cup A)$

4th Normal Form = Multivalued Dependency.

A multi-valued dependency $X \to\to Y$ exists if

$t1[X] = t2[X] = t3[X] = t4[X]$ $\quad Z = R - (X \cup Y)$

then $t1[Y] = t3[Y]$ $\quad t2[Y] = t4[Y]$ Trivial if Y subset of X or $X \cup Y = R$

$t1[Z] = t4[Z]$ $\quad t2[Z] = t3[Z]$

If for all non-trivial multivalued dependency in F

$X \to\to Y$, X is superkey then in 4NF

5th Normal Form = Project-Join Normal Form

Properties of Relational Decomposition

We perform this in an attempt to reduce redundancy of data.

a) Attribute Preservation Condition = Every attribute must appear once

b) Dependency Preservation Property

A decomposition $D = \{R1, \ldots\}$ is dependency preserving if

union of projections of F onto $Ri = F$

$[\pi_{R1}(F1) \cup \pi_{R2}(F2) \ldots]^+ = F^+$

c) Lossless Join property = no spurious tuples due to join => MUST PRESERVE

Algorithm to test for non-additive join property

1st) Initial matrix for each relation Ri having one row and one column j

for each attribute

2nd) Set all to b first

3rd) For each row i

For each column j

if Ri has Aj

make it a

4th) For each FD $X \to Y$ in F → Repeat till whole loop had no change

All rows that have every part of X as a

For all attributes of Y

if any row matched = a

set all to a

contd 4th) This means that every row which has X as a is checked to see if any of the attributes of Y = a and change it in all rows if any such attributes exists

5th) If any row has all a's ∴ Property ✓

else Property X

d) Testing Binary Decompositions

D = {R1, R2} has it if and only if

The F.D $((R1 \cap R2) \rightarrow (R1 - R2))$ in F^+ or

$((R1 \cap R2) \rightarrow (R2 - R1))$ in F^+

e) Successive Non-additive Join

If a decomposition $D = \{R1, \ldots, Rm\}$ and has lossless join. and

$Ri = \{Q1, Q2, \ldots, Qn\}$ has loss less join

∴ $D = \{R1, \ldots, Q1, \ldots, Qn, \ldots, Rm\}$ has lossless join

Synthesis in 3NF

Step 1) Find minimal cover G

Step 2) For each L.H.S X of FD in G

Create relation schema D { {XUA1UA2...} }

where $X \rightarrow A1, X \rightarrow A2, \ldots$ are only in G with X as LHS

Step 3) If none in D has key of R, add one more with key of R

We can get alternate 3NF by starting with same FDs. Due to minimal cover

Synthesis in BCNF

Given R, F = FD and D = R

While there is relation Q in D not in BCNF

do

{

Choose a Q in D not in BCNF, find FD $X \rightarrow Y$ violating BCNF

Replace Q in D by Q − Y and X U Y }

Problems with bad relations

1) Null Values

2) Dangling Tuple = Tuple that does not participate in natural join.