DBMS - 08/01/2020

Data Models - conceptual tools

1. Relational Model

1.1. Uses collection of tables to represent both data and relationships among those data.

1.2. each table has multiple columns and each column has a unique name.

1.3. table is also called relations.

2. Entity Relationship Model

2.1. uses collection of basic objects called entities and relationships among these objects.

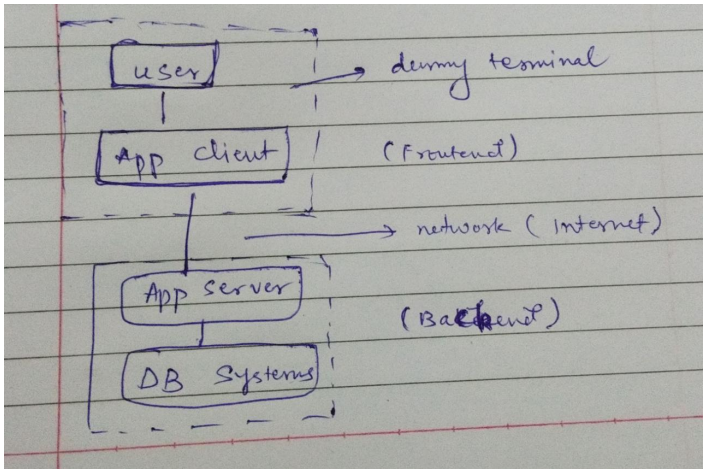
2.2. and entity is a thing or object in the real world that is distinguishable from other objects.

DataBase Architecture

Influenced by the underlying computer system on which the database system runs.

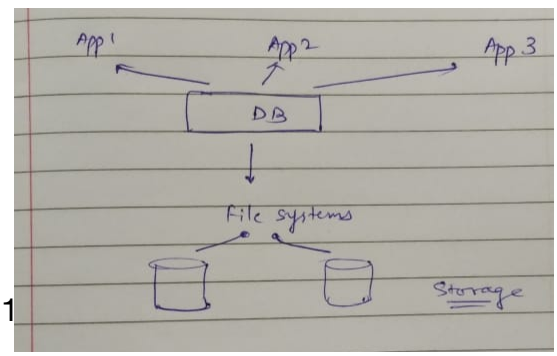
Database systems can be centralised or client-server where one server machine executes work on behalf of multiple client machines.

Three Tier Architecture



User tier (frontend) App server

DB system



DataBase Design

- For a small application, DB designer may decide directly on the relations to be created, their attributes and constraints on the relations.

✦ Initial Phase - characterise fully the data needs of the prospective DB users.

➡ Outcome - user specification document

✦ Designer chooses a data model and translates these requirements into a conceptual schema of the DB

➡ Data model - easy to use and understand by both parties.

➡ The ER model is used to represent the conceptual design of DB

✦ A fully developed conceptual schema indicates the final requirements of the enterprise.

Process of moving from an abstract data model to the implementation of the DB proceeds in 2 final design phases

✦ LOGICAL DESIGN PHASE - mapping conceptual schema into a relational schema

✦ Designer uses the resulting system specific DB schema in the PHYSICAL DESGIN PHASE

Physical features of DB are specification, index, file organisations etc.

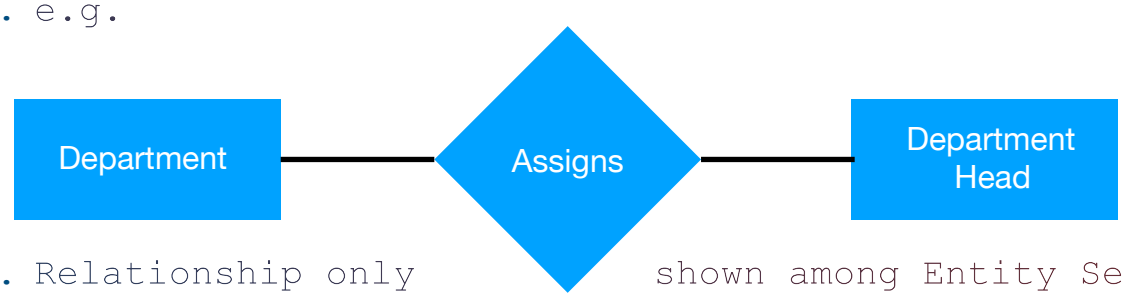
DBMS - 14/01/2020

Major Pitfalls

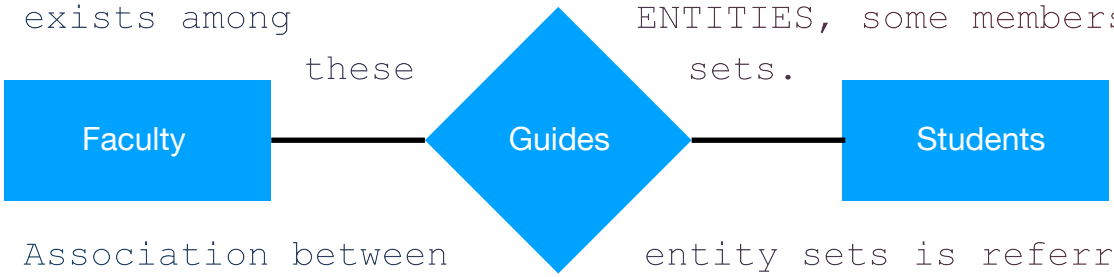
1. Redundancy - same/similar data covered multiple times without adding utility.
2. Incompleteness - a bad design may make certain aspects of the enterprise difficult or impossible to model.

Entity Relationship Model

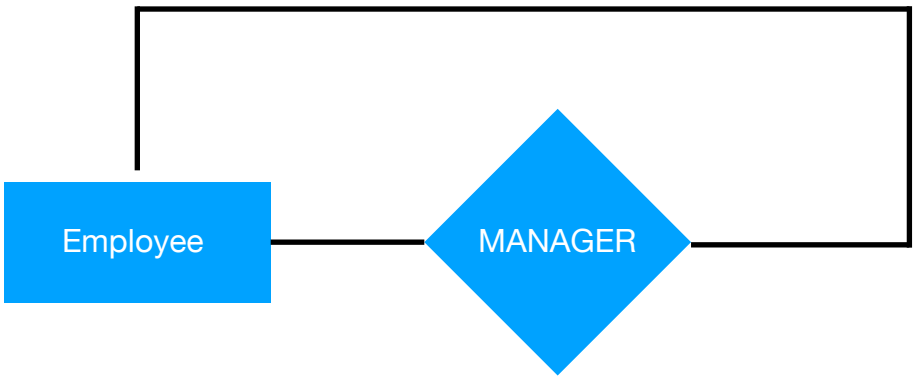
1. Entity set
   1. An entity is a thing or object in the real world that is distinguishable from all other objects.
   2. An entity set is a set of entities of same type that share some properties or **attributes**.
   3. Entity sets need **NOT** be DISJOINT.
   4. An entity is described by a set of attributes. e.g. name, ID, department etc.
2. Relationship Sets
   1. Association among several Entities.
   2. e.g.



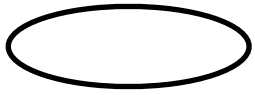
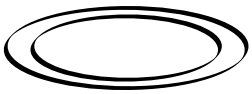
   3. Relationship only exists among these shown among Entity Sets, but ENTITIES, some members of sets.



   4. Association between entity sets is referred to as **PARTICIPATION** into the Relation.
   5. The function that an entity plays in a relationship is called the **Entity Role**. {used in recursive relationships. Roles are pre-defined}
   6. **Recursive Relationship** - some entity set participates in a relationship set more than once, in different roles.

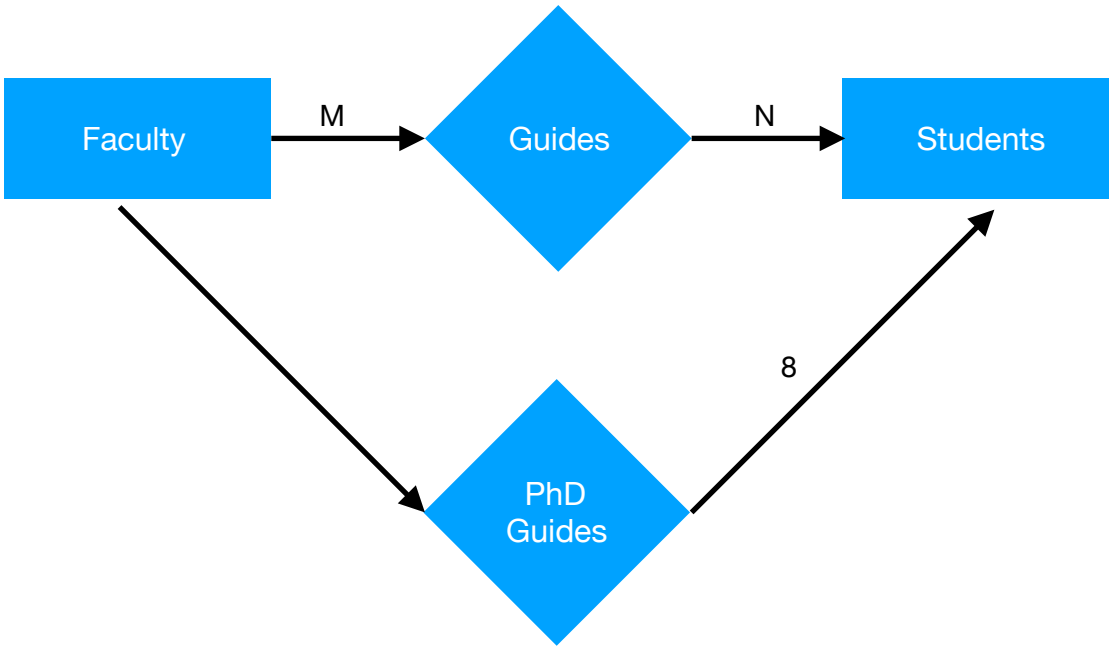


7. A relationship may also have attributes called **Descriptive Attributes.**
   extra information that may be required in accordance with the relationship.
   e.g. for faculty GUIDES student relationship set project name , stipend etc are attributes that may only be used with relationship set, else they lose context. Such attributes are called descriptive attributes for the relationship set.

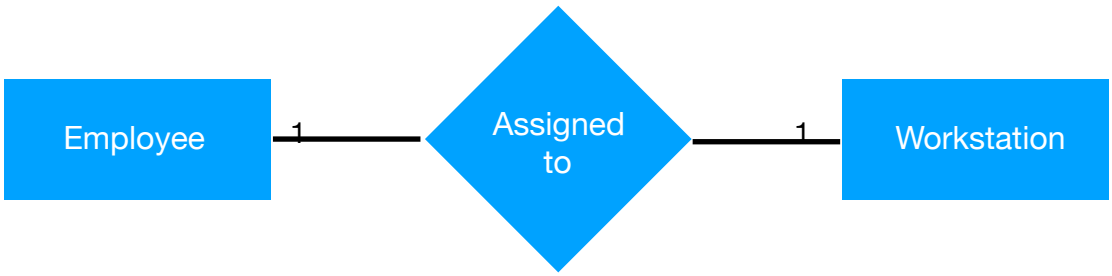
3. Attributes
   1. Single Valued 
   2. Multi Valued 
   3. Derived - may be derived using another attribute e.g. age is derived from D.O.B. every time it is required and never stored. 

4. NULL values
   attributes can have NULL values for some entities
   1. The entity does NOT have a value for the entity
      e.g. middle name
   2. The value is unknown
      1. Missing - the value exists but info is not available
      2. Unknown - do not know if info exits.
5. Simple Attribute - not divided into subparts
6. Composite Attribute - divided into subparts.
7. **CONSTRAINTS** - to which contents of a DB must conform

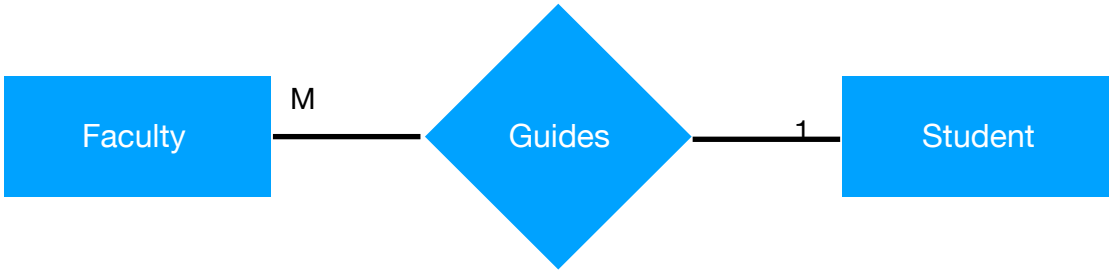
4. Mapping Cardinalities (Cardinality ratio)
   1. Express number of entities to which another entity can be associated with via a relationship set.



   2. Useful in binary relationships
   3. In a binary relationship, mapping cardinalities MUST be one of the following :
      1. One - to - One

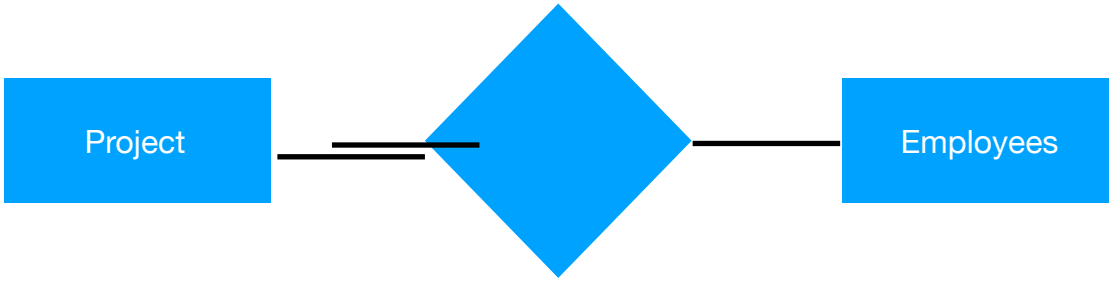


2. One - to - Many
   1. Student guided by one faculty
   2. Faculty may guide multiple students



3. Many - to - One
   1. Many cities in one state
   2. One state has many cities
4. Many - to - Many
   1. Student may be guided by multiple faculty
   2. Faculty may guide multiple students

5. PARTICIPATION
   1. Total
      every entity in the entity set participates in at least ONE relationship in relationship set
      There will be one entry for every entity from entity set in the relationship table



   2. Partial
6. KEYS

1. Key for an Entity is a set of attributes that suffice to distinguish entities from each other

ID	name	DOB	DOJ	PAN	Aadhar
1	abc	..	dmy	a	xxxx
2	..	xyz	dmy	b	xxxx
..	abc	..	..	c	xxxx
Sequencing	..	xyz	dmy	d	xxxx

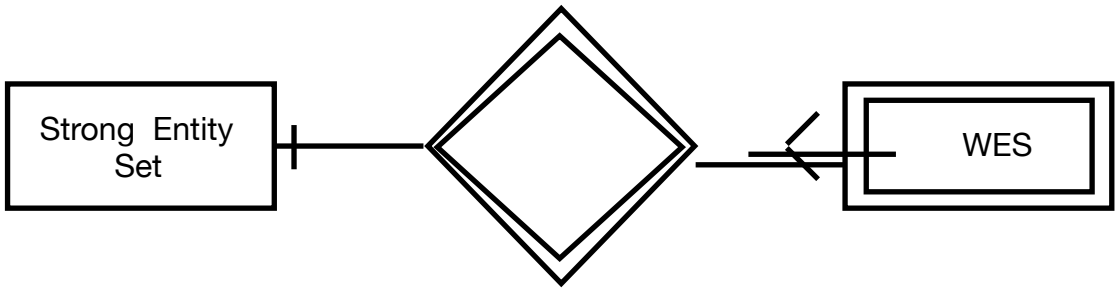
2. Key(s) is(are) chosen such that the attribute is :
    1. UNIQUE
       name, DOB, DOJ - may be same
       ID, Aadhar, PAN etc may be used as primary key
    2. NOT NULL
       phone no. - may be NULL
       ID, Aadhar, PAN may be used - NOT NULL as generated or necessary to possess.
       e.g. Aadhar is mandatory for school admission
3. Four types of KEYs :
    1. **PRIMARY KEY** : attribute used to uniquely identify an entity.
       Underlined attribute(s) in ER Model
    2. **CANDIDATE KEY** : attributes that may be chosen as primary key.
    3. **ALTERNATE KEY** : candidate keys that are NOT primary key.
    4. **FOREIGN KEY** : primary key of another table used to link the two tables.
       may be **Repeated** as well as **NULL**.
       takes values that **exist** in the original table the key refers to.

WEAK ENTITY SET

- Entity set that does not contain sufficient attributes to form a Primary Key

Entities belonging to a weak entity set are identified by being related to specific entities from another entity set in combination with some of their attribute values

- This set is called identifying or owner entity set



- Weak entity set must be associated with an identifying set
- Identifying relationship is many - to - one from weak entity set to the identifying set
- Participation of WES in the relationship is total

e.g.

Course ——————<<>>=========<=SECTION

- Cid — Section ID
- Name — Sem
- Faculty — Year
- Department

(identifying set) (weak entity set)

Section makes no sense without Course to impart context - duplicate entries exist
identifying set is MUST with a WES

Reduction to Relational Schema

- Representation of Strong Entity Set with SIMPLE attributes
- SES => table with attributes as columns
- PKs of ES => PK of relation/ table
- For composite attributes => one column (attribute) for each component
- Derived attributes => NOT stored explicitly
- Multivalued attributes => create new relational schema
  => create multiple columns as per the maximum limit » NULLs in optional fields

Representation of Weak Entity Set

A (WES) = { a1, a2, a3, ... aM }

B (IES) = { b1, b2, b3, ... bN, bI, b J }

Table for A will contain

{ a1, a2, … aM , b1, b2 .. . bN }

All the attributes of A + primary key of B

Representation of Relationship Sets

- R = { a1, a2, a3 … aM } **U** {b1, b2, . . . bN }
- { a1, a2, a3, … aM }
  union of Primary Keys of each of the entity sets participating in R
- {b1, b2, …. bN }
  descriptive attributes

◉ For Binary relationships : Primary Key of the table will be as follows :

- Many - to - Many
  union of Primary Keys of both the Entity Sets
- One - to - One
  Primary Key of **any** of the sets may be chosen
- Many - to - One / One - to - Many
  Primary Key of the ES on the Many side of the relationship set

◉ For n-ary relationship union of PKs from participating ES will serve as the PK

◉ For recursive relationship set
pre - requisite - role indicators associated with the relationship are used as attribute name
the column header is the role in the recursive relation
primary key is the combination of the columns

Identifying Relationship

- No schema is created
- WES has no PK to store in a table
-

Combination of Schemas

- For many - to - one relationship and participation of A (many side) is total
  - Combine A & AB to form a single schema
    e.g. city in states
    the city table schema has a column of state instead of a separate relational table

Relational Algebra and Relational Calculus

Do NOT mix SQL and what follows

- Relational Algebra and Relational Calculus were developed before SQL
- The basic set of **operations** for the formal relational model is the Relational **Algebra**
- These operations enable a user to specify basic retrieval requests as relational algebra expressions
  The result is a new relation
- The relational **Calculus** provides a **higher level declarative language** for specifying relational queries
- In relational Calculus, there is no order of operations to specify how to retrieve the query result – only, **what information the result should contain**
- Relational algebra operations can be divided into two groups :
  - Includes set operations – union, intersection, Cartesian product, set difference
  - Operations specifically for relational DBs – SELECT, PROJECT, JOIN

Unary relation operations

- SELECT (σ)– to choose a subset of tuples (rows) from a relation that specifies the selection condition. $\sigma_{Dno=5}(Emp)$
  - Selection condition is a boolean expression specified on the attribute of relation R
  - Clauses can be connected by standard Boolean operators (and, or, not) to form a general selection condition $\sigma_{Dno=5 \wedge Manager='Smith'}(Emp)$
  - The selection condition is applied independently to each tuple individually
  - Select operator is unary i.e. applied to a single relation
  - **Degree**
    no of columns resulting from a select operator = degree of R
  - The fraction of tuples selected by a selection condition is referred to as the selectivity of condition
  - Select operator is commutative

$\sigma_{<con1>}(\sigma_{<con2>}(R)) = \sigma_{<con2>}(\sigma_{<con1>}(R))$

- PROJECT operator (∏)
  - Select certain columns from the table and discard other columns
  - Removes any duplicate tuples
  - $\Pi_{<name,salary>}(\sigma_{<con1>}(Emp))$
  - If duplicates are not eliminated, result would be a **multi set**
  - Commutativity does NOT hold
    - Table attributes filtered selectively
    - Can't take attribute which doesn't exist

$\Pi_{<name>}(\Pi_{<name,DOB>}(Emp))$ » valid

$\Pi_{<name,DOB>}(\Pi_{<name>}(Emp))$ » NOT valid » DOB doesNOT exist in inner ∏

Select used to filter rows (compare, test and show rows) » full tuples shown

Project used to filter columns » all rows shown

Both used together to obtain sub-table

- In SQL, PROJECT attribute list is specified in the SELECT clause of the query
- If we remove the keyword DISTINCT from SQL query, duplicates will NOT be eliminated
- SELECT DISTINCT Salary from Emp

Inline expressions

$\Pi_{<Fname,Lname,salary>}(\sigma_{Dno=5}(Emp))$

OR

- Explicitly show sequence of operations, giving a name to each intermediate relation and using using assignment operations (<——)
- Demp5 <——— $\sigma_{Dno=5}(Emp)$
- Result (first_name, last_name, salary_of_emp) <——— $\Pi_{Fname,Lname,salary}(Demp5)$
- Rename Operator (ρ)
- $\rho_{S(B1,B2...Bn)}(R)$ s : new relation name ; Bi : new attribute names
- For SQL : use KeyWord AS.
  - SELECT E.fname AS first-name FROM Emp AS E WHERE Dno=5

Relational Algebra operations from Set Theory

- UNION, INTERSECTION, SET DIFF
- Binary operations

- Union Compatibility

  2 relations MUST have the same type of tuples

  - Number of columns should be same
  - Data types of each corresponding column should be same

e.g. Emp (EID <int> ,Ename <varchar>, DOJ <date>, PAN <varchar>)

Is Union compatible with

Student (SID <int> ,Sname <varchar>, DOB <date>, PAN <varchar>)

UNION R **U** S

Includes tuples that are either in R or in S or in both R and S

Duplicates are eliminated

First tuples or R are shown, then S

INTERSECTION R <intersection> S

Includes tuples in both relations

SET DIFF (R-S)

All rows that are present in R but NOT S

Cartesian product (cross product) – cross Join (**X**)

Emp (EID <int> ,Ename <varchar>, DOJ <date>, PAN <varchar>)

X

Student (SID <int> ,Sname <varchar>, DOB <date>, PAN <varchar>)

=

(EID <int> ,Ename <varchar>, DOJ <date>, PAN <varchar>, SID <int>, Sname <varchar>, DOB <date>, PAN <varchar>)

Emp{ E1, E2, E3 … Em } **X** Student { S1, S2, S3 . . . Sn } = { E1S1, E1S2 , … E1Sn , E2S1, E2S2, … E2Sn… EmSn }

22/01/2020

JOIN ( ) ⋈

- Combines related tuples from 2 relations into a single longer tuple
- Join ~= cartesian product + selection condition
  - First take a cross product and then filter the product with some selection criterion
- e.g. Emp ⋈ Dept (Emp.Dno = Dept.Dno)
  -Cartesian product of Emp and Dept with only the tuples where Emp.Dno=Dept.Dno
- e.g. Emp ⋈ Dept (Emp.Dno = Dept.Dno & Emp.salary ≥ 1000)
  -Cartesian product of Emp and Dept with only the tuples where Emp.Dno=Dept.Dno and Emp.salary ≥ 1000
- Result of Join is a Relation with n+m attributes if R(n) and S(m)
- Q has one tuple for each combination of tuples
- A Join with such a general join condition is called Theta Join
- Tuples whose join attributes are NULL or for which the join condition is false do not appear in the result

EQUIJOIN or Natural JOIN

- Only = is used
- Have one or more pairs of attributes that have identical values in every tuple
- since, one of each pair of attributes with identical values is superfluous,
  - Natural Join = EquiJoin + removal of superfluous attribute
  - Natural Join requires that the two join attributes have the same name in both relations
- Join selectivity –> ratio of **expected size of Join result** with the Max size n*m

Inner Join

Match and combine operation defined as a combination of Cartesian product and selection

R (join with condition) = $\sigma_{<cond>}(RXS)$

Division operation

Result of division is a relation T(Y) that includes a tuple t if tuples t_k appear in R with $t_R[Y] = t$ and with $t_R[X] = t_s$ $\forall t_s$ € S

Retrieve names of employees who work on all projects that John Smith works on

- SMITH <— $\sigma_{Fname='John' Lname='Smith'}(Emp)$
- SMITH_PNOS <— Π_{PNo}(WORKS_ON $\bowtie_{Empno=Eno}$ SMITH)

- Emp_Smith(Eno) <— WORKS_ON ÷ SMITH_PNOS
- Result <— $\Pi_{Fname,Lname}$(Emp_Smith*Emp)
- Used in SQL as EXISTS
  - SQL EXISTS
    - TRUE
    - FALSE

28/01/2020

Generalized Projection

- Extends projection operation by allowing functions of attributes to be included in the projection list
- $\Pi_{ename,sal}(employee)$ <– projection
  $\Pi_{ename,sal-0.05*sal}(employee)$ <– generalized projection

Aggregate functions

- SUM, AVERAGE, MAXIMUM, MINIMUM etc.
- Used with Projection(s) / Selection / where conditions

Q ———————————————————————————————

Statement to display department name and department numbers such that average salary in the department ≥ 50k

Two tables : Emp (Eno, Ename, Dno, Sal) ; Dept (Dno, Dname, loc)

———————————————————————————————

Grouping

- Grouping tuples in a relation by the value of some of their attributes and then applying an aggregation function independently to each group
- $_{<groupingattribute>}$<grouping symbol>$_{<functionlist>}(Relation)$ ב
  $\rho_{R(Dno,noofemp,avgsal)}(_{Dno}\mathcal{G}_{count(eid),Average(sal)}(Relation))$
- In general, duplicates are not eliminated when an aggregate function is applied
- NULL values are not considered in the aggregation

Recursive Closure operations

- Applied to recursive relationship between tuples of same type
  e.g. relationship between employee and manager

Q

Retrieve all reports of an employee at all levels i.e. all employees e' directly supervised by e, all employees e'' directly supervised by e' and so on

- easy to specify all employees supervise by e at a specific level by joining table with itself one or more times

- However, difficult to specify all supervisors at all levels
- JamesID <— $\Pi_{Id}(\sigma_{name='James'}(emp))$
- Supervision(eid1,eid2) <—$\Pi_{eid,sid}(emp)$
- Result(eid) <— $\Pi_{eid1}(Supervision \bowtie_{sid=eid} JamesID)$
- Result2(eid) <— $\Pi_{eid1}(Supervision \bowtie_{eid=eid} Result)$

29/01/2020

Outer Join Operation(s)

* Join operation matches tuples that satisfy the join condition
* Types of join where tuples (rows) with NO match are eliminated is called **Inner Join**
* **Outer Join –>** user wants to keep all the tuples in R, or all in S, or all in both of these relations in the result of JOIN, regardless whether they have matching tuples in the other relation
* Left Outer Join — ⟕
* Right Outer Join — ⟖

* List all employee names as well as names of the department they manage if they happen to manage a department, if they do not manage one, we can indicate with a NULL value
* $\Pi_{empname,deptname}(emp ⟕_{eid=mngrid} dept)$

Outer Union Operation

* To take union of tuples from 2 relations that have some common attributes, but are NOT union compatible
* **R(X,Y) & S(X,Z)**
* Tuples are partially compatible
* Attributes that are union compatible are represented only once in the result, and those attributes that are not compatible from either relation are also kept in the result relation **T(X Y Z)**

Structured Query Language — SQL

* non-procedural language
* Classified as :

- ➡ Data Definition Language DDL
  - Schema related
  - Auto Commit – cannot be undone
    - Create
    - Alter
    - Drop
    - Truncate - delete all data in table but keep schema
- ➡ Data Modification Language DML
  - Data related
    - Insert
    - Update
    - Delete - can be undone using Rollback
- ➡ Data Control Language DCL
- ➡ Transaction Control Language TCL

04/02/2020

NORMALIZATION

1. Process of efficiently organising data in a DataBase
2. Two - step process :
    1. Put data in tabular form by removing **repeating groups**.
    2. Remove **duplicate data** from the relational tables.

Anomalies :

i) Insertion anomaly
    i) A set of columns NULL, because the data is not assigned, NOT because data is not available.
    ii) E.g. for Employee table : ProjectID, ProjectName etc all fields related to Project are NULL till a project is allotted.
    iii) Different from NULL value if data is not available (say contact is not provided)
    iv) An insertion anomaly is the inability to add data to the database due to absence of other data. For example, assume Student_Group is defined so that null values are not allowed. If a new employee is hired but not immediately assigned to a Student_Group then this employee could not be entered into the database. This results in database inconsistencies due to omission.

ii) Updation anomaly
    i) Numerous instances with the same value that need updation
    ii) e.g. Bombay changed to Mumbai : all instances where Bombay used as a place need to be changed to Mumbai
    iii) An update anomaly is a data inconsistency that results from data redundancy and a partial update. e.g. if A. Bruchs' department is changed, it needs to be updated at least twice.

iii) Deletion anomaly
    i) A deletion anomaly is the unintended loss of data due to deletion of other data. For example, if the student group Beta Alpha Psi disbanded and was deleted from the table above, J. Longfellow and the Accounting department would cease to exist. This results in database inconsistencies

and is an example of how combining information that does not really belong together into one table can cause problems.

Employee_ID	Name	Department	Student_Group
123	J. Longfellow	Accounting	Beta Alpha Psi
234	B. Rech	Marketing	Marketing Club
234	B. Rech	Marketing	Management Club
456	A. Bruchs	CIS	Technology Org.
456	A. Bruchs	CIS	Beta Alpha Psi

Normal Forms

A table is said to be in a normal form if it satisfies a certain set of Constraints

SIX normal forms :

1) First Normal Form (1NF)
2) Second Normal Form (2NF)
3) Third Normal Form (3NF)
4) BCNF
5) Fourth Normal Form (4NF)
6) Fifth Normal Form (5NF)

e.g.

itemNo |item_desc |price |MfgName |distrName |orderID |date | Qty

First Normal Form

- NO repeating groups
  - No column with
    - Multiple values (ONLY one value in one cell)
    - Composite values (only simple values in cell e.g. address not in one cell, keep different cells for houseNo, city, PIN etc.)
- All attributes are dependent on the Primary Key

PID | Pname | loc | Rev | Mgrid | ClientID

All attributes dependent on Primary Key :

Select * from Project where PID='P01'

the output is a **single** row with ALL attributes

If we know the PID, we know value of every attribute

Second Normal Form

- a table is in First Normal Form
- All non-key Attributes are fully functionally dependent on ALL the candidate Keys
- Non - key attributes : attributes NOT part of ANY candidate key
- Fully Functionally dependent : for a given relation R, attribute B of R is fully functionally dependent on attribute A of R, if it is functionally dependent on A and NOT on any Subset of A
- Functional dependency : A —> B ; RollNo —> Name
- Table Order_details

Oid	PrId	PrName	Mfg date	Price
01	p1	soap		
01	p10	shampoo		
01	p26	bread		

PrName — Non Key attribute

dependent on PrId which is a subset of primary key

Mfg date , Price — dependent on the complete primary key

Transaction

CNo | Cname | Ccity | Cphone | PNo | TimeStamp | Qty

CNo —> Cname, Ccity, CPhone (partial dependency on primary key)

CNo, PNo, TimeStamp —> all attributes

CNo | PNo | TimeStamp | Qty CNo | Cname | Ccity | Cphone

Transitive Dependency

- value in a no - key field is determined by value in another non - key field and that field is not a part of any candidate key
- (a, b), (b, c) $\Longrightarrow$(a, c)
- A (non-key) —> B (non-key)
- PK—>A ; PK—>B

ProjectNo | ProjectTitle | ProjectManager | Phone

ProjectManager —> Phone
ProjectNo —> ProjectTitle, ProjectManager, Phone

Third Normal Form

- Table should be in Second Normal Form
- All attributes are dependent ONLY on the Candidate Key(s)
  i.e. No Transitive Dependency
- All non - key attributes should be mutually independent
- To transform a table into 3NF : split the table with dependent non-key attributes in new table

Questions on normal form done in next class

18/02/2020

Integrity Constraints

- Not Null
- Unique
- Primary key (not null + unique are automatic)
- Default - keep attributes with default values to the right of the table
- Check - the input should be from the specified set of values | works as enum
- Create table emp
  (name varchar NOT NULL,
  PAN varchar UNIQUE,
  isActive varchar DEFAULT 'Y' check in ('Y','N'))

Referential Integrity

- Applicable in case of foreign keys
- Foreign key refers to primary key of another table
- The value in the foreign key MUST be a value as the primary key in reference table or NULL
- If the reference is deleted
  - The FK may be assigned NULL or some other apt value
  - The records with deleted FK value may be deleted
- The updation is situation based but Must ensure Referential Integrity

Triggers

e.g. a value change must be reflected wherever referenced instantaneously

- Create Trigger - used to implement actions in SQL

  e.g. Create Trigger <Trigger_name>

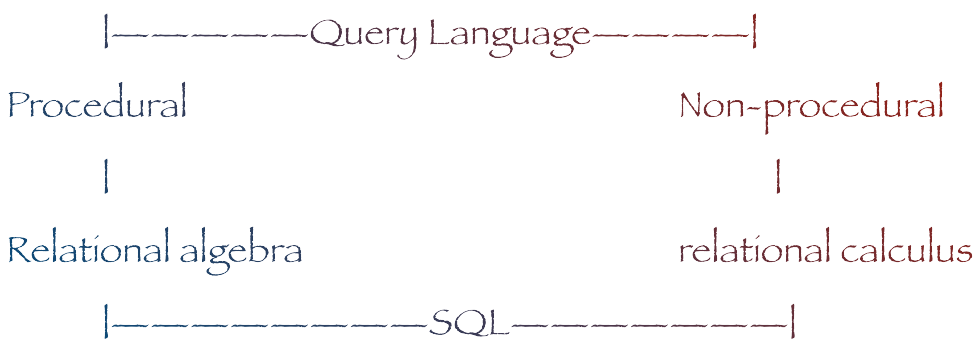
  Before | After Insert or Update of Salary, Supervisor_ID on emp

  Update Salary =0 where Supervisor_ID =134

- Trigger is regarded as Event-Condition-Action (ECA)
  - Event(s) - usually DB update ops that are explicitly applied.
    - Specified after keyword BEFORE | AFTER
  - Condition - determines whether the rule action should be executed
    - Once the triggering event has occurred, an optional condition may be evaluated. If no condition is specified, action will be executed once the event occurs.
    - If a condition is specified, it is evaluated first and only if it evaluates to true will the rule action be executed.
    - This condition is specified in the WHEN clause of the trigger
  - Action - usually a sequence of SQL statements

Tuple Relational Calculus

Tuple ≡ row of the table



Relational Calculus

1. Tuple relational calculus - row wise
2. Domain relational calculus column wise

- User is concerned with the details of how to obtain the end results
- Uses mathematical predicate calculus
- A predicate is a truth-valued function with arguments
- When we replace with values for the arguments, the function yields an expression called a proposition, which will be either true or false.

Tuple Relational Calculus

- write ONE declarative expression to specify the retrieval request
- The expressive power of relational algebra and relational calculus is IDENTICAL the same results can be obtained using either of them

Tuple variables and range relations

- TRC is based on specifying a number of tuple variables
- Each tuple variable usually ranges over a particular DB relation.
- The variable may take as its value any individual tuple from that relation
- { t | COND (t) }

  e.g. {t|EMP(t)} $\Longrightarrow$ t spans over the EMP & may take any value from EMP

  { t.empname, t.empno | emp(t) AND t.sal >50000 }
- t.sal ≡ t[sal] ≡ value in sal column of row represented by t
- The following need to be specified in a TRC expression :
  - To each tuple variable t, the range relation R of t : R(t)
  - A condition to select a particular combination of tuples
  - Requested attribute(s)