DATABASE MANAGEMENT SYSTEM

Module-I

Database system architecture; Data Abstraction, Data Independence, Three-Schema Architecture Data models: Entity-relationship model, network model, relational and object oriented data models, integrity constraints.

Module-II

Relation Query Languages: Relational Algebra, SQL, Data Definition Language (DDL), Data Manipulation Language (DML), Tuple and Domain Relational Calculus.
Relational Database design: Domain and data dependency, Armstrong's axioms, Normal forms, Dependency preservation, Lossless design.

Module-III

Query processing and optimization: Evaluation of Relational Algebra Expression, Query Equivalence, Join strategies, Query optimization Algorithms.

Module-IV

Transaction processing: Concurrency control, ACID property, Serializability of scheduling, Locking and time stamp based schedulers, Multi-version and optimistic concurrency control schemes, Database recovery.

Module-V

Advanced Topics: (Introduction to concepts only) Object oriented ~~data~~ and object relational databases, Logical databases, Distributed databases, Data warehousing and data mining.

L-1 (27.07.2023) (11:00-12:00) DBMS

Objectives:
→ Basic concepts, modeling, architecture
→ Design aspects of database
→ Use of DDL & DML for query purpose
→ Query evaluation and optimization
→ Basic issues & concept - Transaction management.

- Remembering
- Understanding
- Applying
- Evaluating
& - Creating

(Bloom's Taxonomy)

* State of inconsistency, failed transaction, recovery has to be done.
The log file records each activity. Helps in recovery management.
* Redundant Data: data being referred in many places/multiple places.
* Vendors of Database: MySQL, oracle, MS SQL, MongoDB Atlas.
DBMS is an application software that facilitates storage, access & different activities with the DB.

[R] Textbook: 6e or 7e Silberschatz, Korth, Sudarshan (Database system concepts, 7e) ↳ www.db-book.com
[A] Enroll IIT KGP swayam online course - Partha Pritam Das.

⊙ Railway reservation system: Robust Database system
· shopping, social media platforms: use multimedia data (no specific structure)
↳ NoSQL (RDBMS is not sufficient) ↳ Referred to as Big Data
↳ Data capturing & handling, similar to XML/HTML (use of tags)
↳ Modern Database Applications.

File system DB
↳ Data Redundancy
↳ Inconsistency
↳ Difficulty in access

L-2 (28.07.2023) (11:00 - 12:00 p.m)

⊙ Database Management System is an application software that implements database and its applications.

⊙ 2-tier and 3-tier architecture
↓ (or client, application layer, db layer)

Client/user set on the frontend side
Data storage & operations - backend (application server)

Data Models:

We use data models to describe certain levels of data abstractions.

→ Data models are the conceptual tools, used to describe certain levels of data abstraction
↳ high level description of data.

Represent: Data, Data relationships, Data semantics, Data constraints

→ Representational Data Models: no detailed physical description required
→ Physical Data Model: physical description of data.

- Relational model
- Entity-Relationship model (ER-model)
- object-based data model (object-oriented & object-relational)
- semi-structured data model (XML)

Older Data Models:
- Network model (form of a graph)
- Hierarchical model (form of a tree)

ER-model
- provides concepts of entity, relationships & attributes.
  ↳ identifying characteristics.

→ Entities:
→ course
→ student/Faculty
→ classroom
→ Library
→ Timetable
→ Department
→ Hostels
→ Section

- E.g: University Database
- Relationship: interaction among student & enrolled course

Relational Data Model:

Ted Codd: Turing Award 1981

↳ All the data is stored in various tables.

	columns: ID	name	dept_name	salary
t1	222	X	Physics	50,000
t2	213	Y	Finance	95,000

(rows)

• The instructor Table

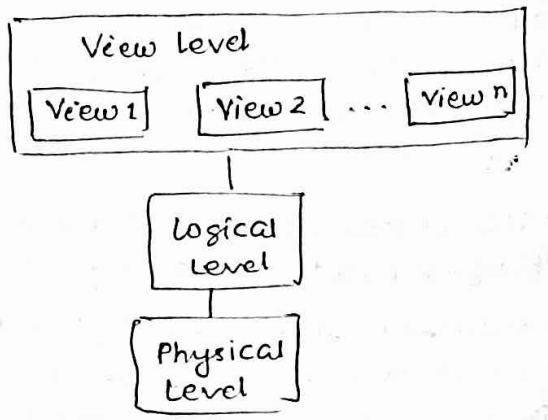
- Each row is called as Tuple/Instance
- columns represent attributes.

Instances and Schemas:

Instructor (ID, name, dept_name, salary)
↳ is a schema
↳ dept (dept_name, building, budget)

Schema is analogous to variable
Instance is analogous to value assigned to the variable.

Isolated access for different kind of users.
(conceptual level)



→ In database, data and schema are stored separately.
→ Schema refers to table name, attribute name, data types and constraints.
- character as well as number - VARCHAR
- numbers, strings ..

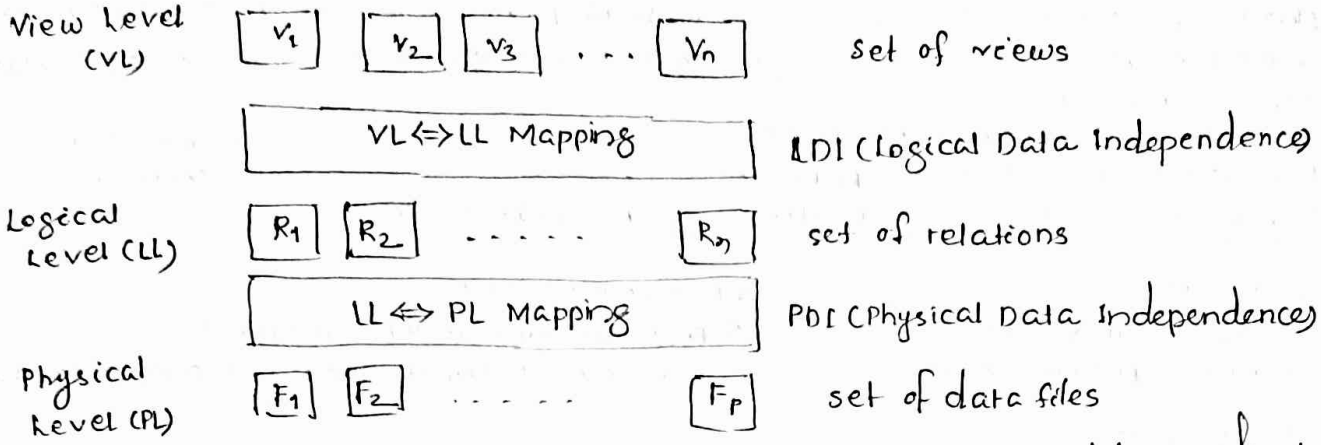
• Primary key, Foreign key are specified as constraints while desining the schema.

• Database Definition:

↳ setting up the skeletal structure

• Database loading/Populating:

↳ Storing data



• The whole database is not projected to all. A portion is projected to individual view.

• L-3 (03.05.2023) (11:00-12:00 p.m.) (12:00-12:30 p.m.)

• The development process of a database system: 3 level process / step

step:01 Gathering/Collecting the requirements.

step:02 Conversion of data models into representational model (ER/object-oriented/semi-structured) (Identification of model)

step:03 Write the application program using general purpose programming language.

⊙ In database perspective, requirements are of two types:

(a) Data model requirements

(b) Functional requirements

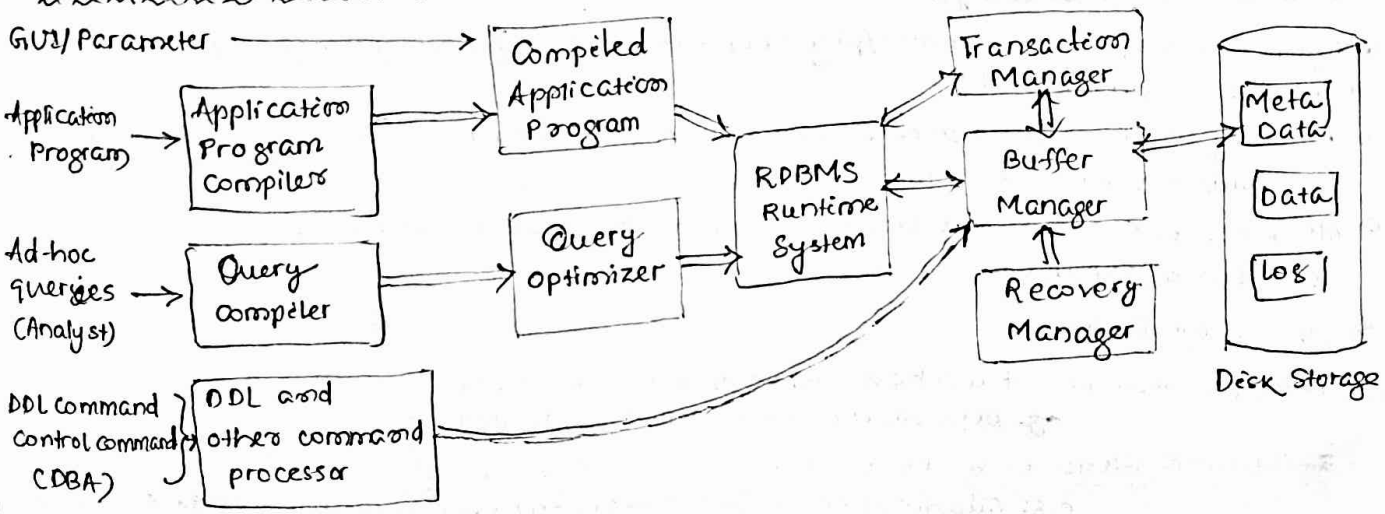
embedded into SQL Queries

⊙ Through the application program, we access the database values, through some variables.

⊙ Embedded SQL, Using SQL commands within the high level language to interact with the database and carry out necessary functionalities.

[B] Navathe Book

• Architecture of RDBMS System:



⊙ Data dictionary contains definition of table, mappings & schema.
⊙ Meta data stores / contains the data dictionary.
⊙ Log contains activity record, old value as well as new values for any change in database value.
↳ used for recovery purpose.

⊙ DDL : Data Definition Language, DBA: Database Administrator
Commands for handling authorization is also controlled by DBA.

⊙ Query optimizer selects the appropriate execution plan. Also optimizes join operations.

⊙ Application program compiler separates the embedded SQL commands. Application program is compiled separately. SQL commands are compiled. And then both are integrated. (3 tasks)

⊙ Transaction Manager
- Tracks the start & end of a transaction
- Records the changes in log
- Emphasizes concurrency control (enforces) Protocols
  - Time stamp Protocol

(Process synchronization. - OS)

⊙ Buffer Manager
↳ manages the disk space.
↳ implements paging mechanism.

⊙ Recovery manager
↳ Performs some undo/redo operation.
↳ Takes control, brings system to a consistent state.

[8] - 2 tier vs 3 tier..
Levels of abstraction..

* Different User Roles:

⊙ DBA (Database Administrator)
- configure database
- write DDLs
- physical layout
- creating data dictionary
- design logical schema
- grant & revoke permissions

(use command line interfaces)

⊙ Data Entry Operators
+ Provided with GUI through which they interact with DB.
→ Feed the data and invoke particular operations.

⊙ Sophisticated User
→ Analysts
→ specify queries to get the data (SQL Query)

* ER Model:
→ Proposed by Peter P. Chan (1970)
→ Trying to establish relationship between the entities.
→ Graphical representation: ER-Diagram
ER-Diagram —convert into→ Schema

→ Try to identify the attributes for individual entity.
(what describes that entity).

e.g: Department Data Base
⊙ Entities: Department, Faculty, Student, Course, Enrollment..
⊙ Relations: Faculty offers a course (teaching)
Student belongs to department.
Student enrolled in a course.

⊙ Entity is a thing of independent/physical existence, which is distinguishable. Individual database contexts.

⊙ An entity set is a collection of entities carrying all same properties (characteristics).
e.g: Student entity set is a collection of all student entities.

⊙ Attribute: Entity set is described by attribute, that carries some value.
e.g: Roll no. in STUDENT entity set.

⊙ Types of attributes:
1. Simple attribute: it will have atomic or indivisible values.
e.g: Department name cannot be divided.
2. Composite Attribute: We have several components in the value
e.g: Qualification — (start year, end year, institution, name of degree)
↳ the subparts of qualification

3. Derived attribute: value depends on some other attribute value.
   - e.g: age - can be derived from DOB.

4. Single-valued attribute: Some attributes can only take a single value.
   - e.g: Place of birth.

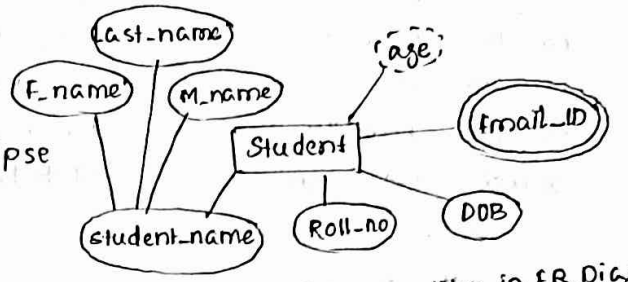
5. Multi-valued attr: (email ID may have more than one values)

- simple single-valued attribute, simple multi-valued attribute
- composite single-valued attribute, composite multi-valued attribute

@How exactly the types of attributes are represented? : Diagramatic Notation

→ Entity - Rectangle
→ Attribute - Ellipse connected to rectangle
→ Multivalued attributes - double ellipse
→ composite attributes - ellipse connected to ellipse
→ Derived attributes - dashed ellipse

Example: Entity: Student



@Representation of an entity in ER Diagram.

* L-4 (04.08.2023) (11:00-12:00 p.m)

& Domain of Attributes:

- Each attribute takes the value from a set called 'domain'.
- e.g: age domain of students.
- Domain of composite attribute is the cross-product of domains of sub-attributes.
- Domain of multivalue attributes is the set of sub-set values from the basic domain.

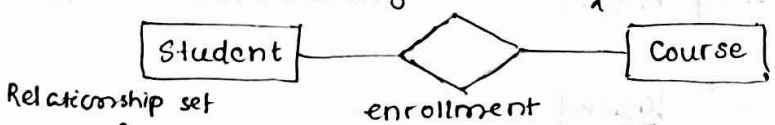
& What is a key:

- Key is an attribute or combination of attributes which is used to uniquely identify a particular tuple of an entity set.
- For a particular entity set, we have more than one key(s) possible.
- Candidate key set: set of keys.
- The minimal candidate key is referred to as the 'primary key.'

{ designer evaluates meaning of individual attributes
  forms the candidate key set, finds the minimal of the keys: primary key.

& Relationship:

- The association among the entities (entity sets) is referred to as relationship.



(Diamond: represents relation)

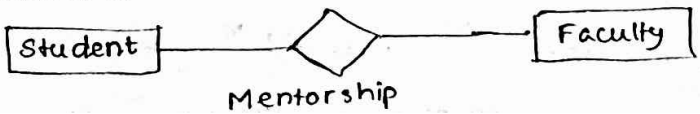
⊙ enrollment ⊆ student × Course
⊙ (s,c) ⊆ enrollment i.e, student s has enrolled course c.

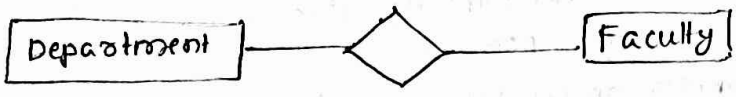
& Degree of Relationship:

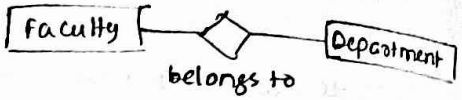
(1) Binary Relationship
(2) Ternary Relationship
(3) n-ary Relationship

- We say the no. of participating entities as degree.
- In ER-model, binary relationship is mostly used.

& Examples of binary Relationships:





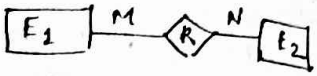


- Department has faculty
- Faculty belongs to the department

Entities		Relationship
Student	Course	enrollment
Student	Department	belongs to
Faculty	Department	belongs to
Faculty	Course	offers / teaching
Student	Faculty	is mentored by / mentorship
Course	Classroom	alloted / assigned

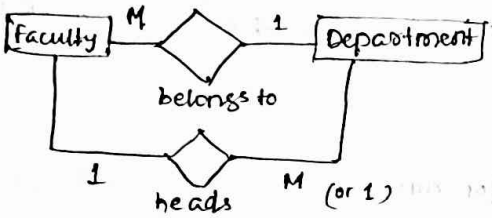
⊙ Cardinality Ratio

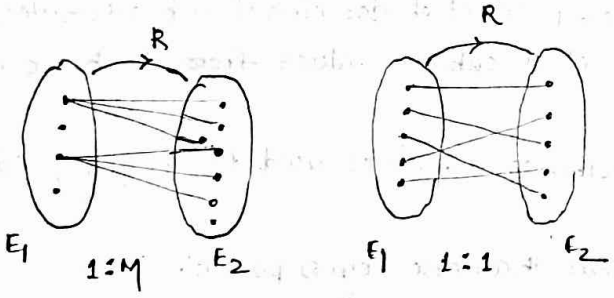
(Set & Relations)



→ cardinality ratio in binary relationship has four possibilities:

- 1 : 1 (one-one)
- 1 : M (one-many)
- M : 1 (many-one)
- M : N (many-many)



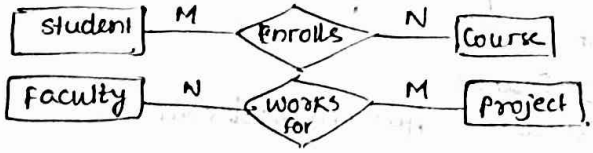


↓ Examples:

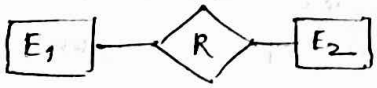
One-to-one:

Many-to-one / one-to-many:

Many-to-many:



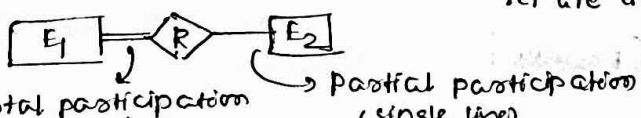
↓ Participation constraints in a Relation:



1. T.P. (Total participation)
2. P.P. (Partial participation)

⊙ In case of total participation all the entity sets should participate in the relationship.

⊙ In partial participation, not all entities in the set are involved in the association.



e.g.

(a particular dept. may not have a faculty)

07

1-5 (05.08.2023) (10:00 - 11:00 a.m)

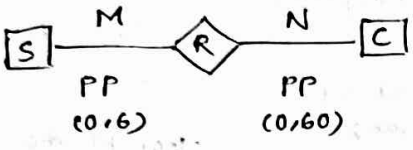
↓ Structural constraints:
→ Cardinality Ratio
→ Participation constraint

• Min-Max: (M,n)

→ Need to be specified along the line from entity to relationship.
→ 'm' is the minimum number of times a particular entity must appear in the relationship at any point of time.
. In case of partial participation, m = 0
. But in total participation, m ≥ 1

→ 'n' is the maximum number of times a particular entity can appear in the relationship tuple at any point of time.

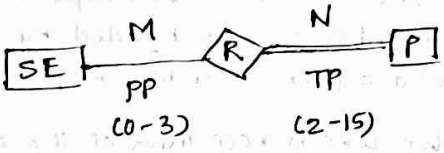
e.g:



. All the students needn't register the course, but they can go maximum upto 6 (six) courses
. All the courses needn't be registered by a student, if at all they register, maximum 60 students can take up the course.

e.g: Each and every project requires atleast two members and maximum of fifteen members. All the software engineers needn't be the part of project, if they participate, they can participate in maximum three projects. Find the type of relation and cardinality ratio.

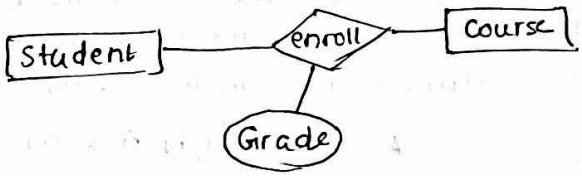
→ It is a binary relationship. with cardinality ratio M:N (many-to-many)



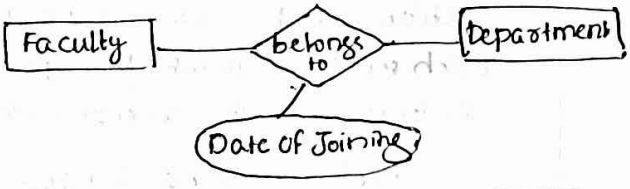
• Attributes for Relationship Type:

A relationship may have some attributes).

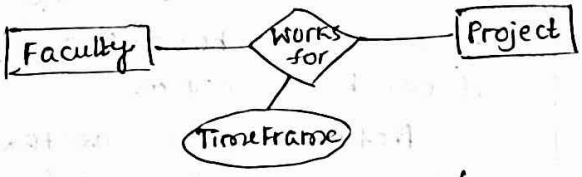
ex 1



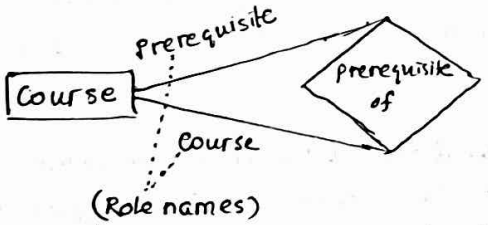
ex 2



ex 3



↳ Recursive Relationship:



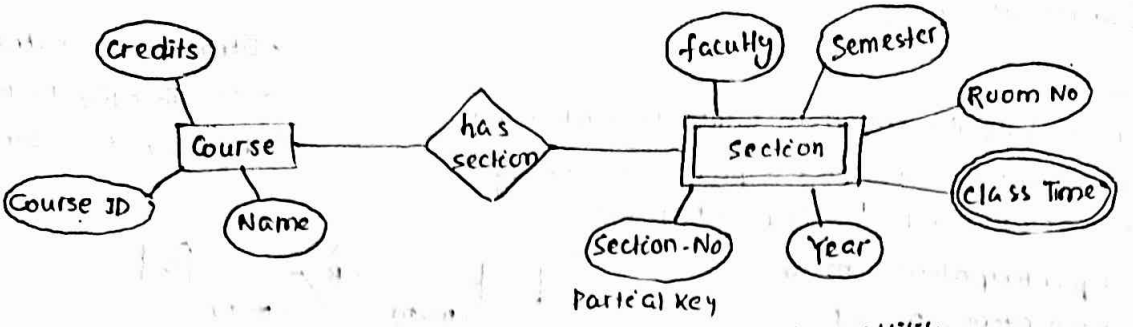
↳ used to specify the exact roles in which the entity participates in the relationships
↳ are essential in recursive relationships
↳ are not essential in non-recursive relationships

↳ Entity Sets:
- Strong entity sets — e.g: Student — strong
- Weak entity sets — Section — weak

• Entities having uniquely identifying attribute } strong entity set
↳ own attribute will be used as key value

• Attributes of entity are not sufficient for identifying entity sets uniquely } weak entity set
↳ The members are dependent on existence of strong entity set i.e. weak entity set doesn't have any independent existence.

• DBMS - Prerequisites
→ Linear Algebra
→ set Theory
→ OSA
→ Basic Programming



⊙ To uniquely identify tuples of weak entity set, primary key of the strong entity is used with the partial key.

• e.g: Equipment (strong), Usage (weak), Relationship: Utility

⊙ Added intentionally to a relation to uniquely identify tuples in the relation — Discriminator. (in weak entity sets)
(e.g: token number for transactions)

⊙ Discriminator: combination of primary key & partial key
(strong) (weak)

• Real-time simulation as well as emulation
OPAL-RT system
(70-80 lacs)

Clear ER-diagram
* Try to create entity set, relationships, specify cardinality ratio (str. constraints)

& Complete Example for E/R schema:

Data model requirements are given.

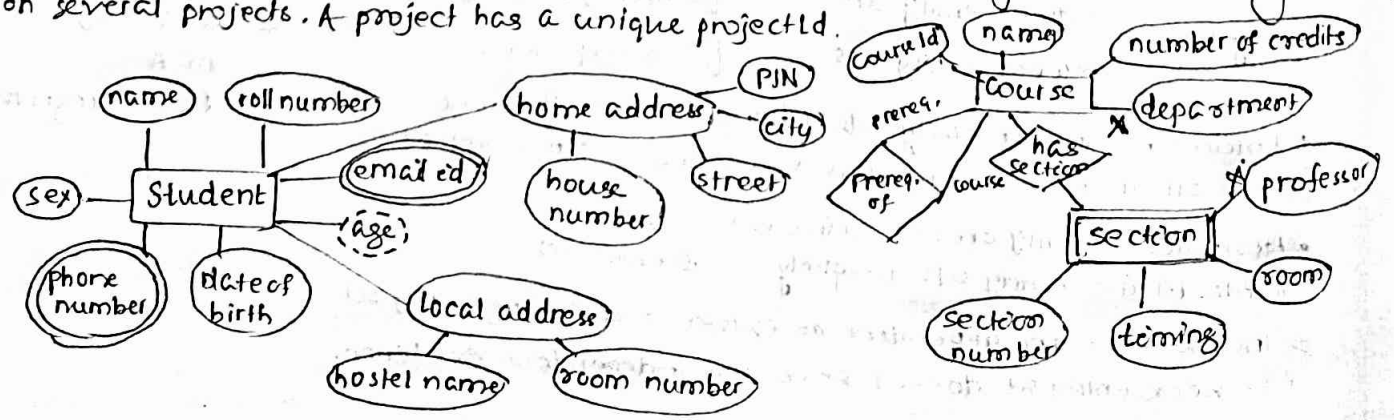
Specification: In an educational institute, there are several departments and each student belongs to one of them. Each department has a unique department number, a name, a location, phone number and is headed by a professor. Professors have a unique employee id, name and a phone number. A professor works for exactly one department.

We like to keep track of the following details regarding students: name, unique roll number, sex, phone number, date of birth, age and one or more email addresses. Students have a local address consisting of the hostel name and the room number. They also have home address consisting of house number, street, city and PIN. It is assumed that all students reside in the hostels.

A course taught in a semester of the year is called a section. There can be several sections of the same course in a semester; these are identified by the section number. Each section is taught by a professor and has its own timings and a room to meet. Students enroll for several sections in a semester.

Each course has a name, number of credits and the department that offers it. A course may have other courses as pre-requisites i.e., courses to be completed before it can be enrolled in.

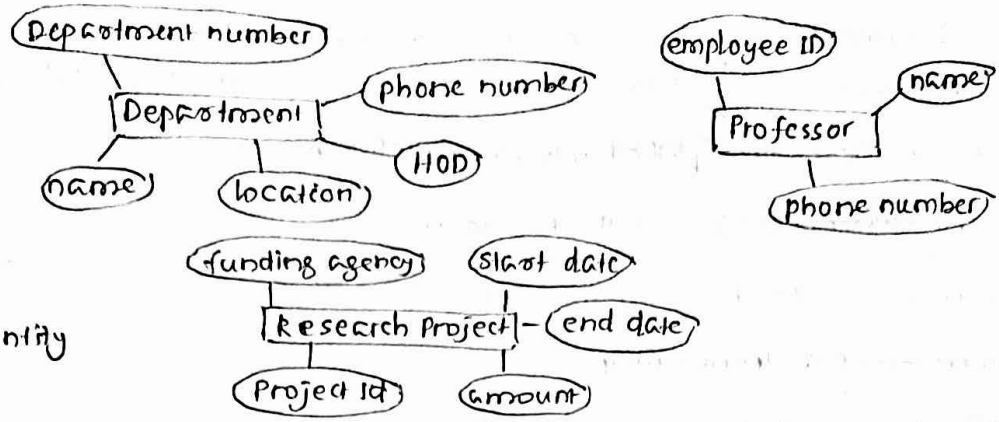
Professors also undertake research projects. These are sponsored by funding agencies and have a specific start date, end date and amount of money given. More than one professor can be involved in a project. Also a professor may be simultaneously working on several projects. A project has a unique projectId.



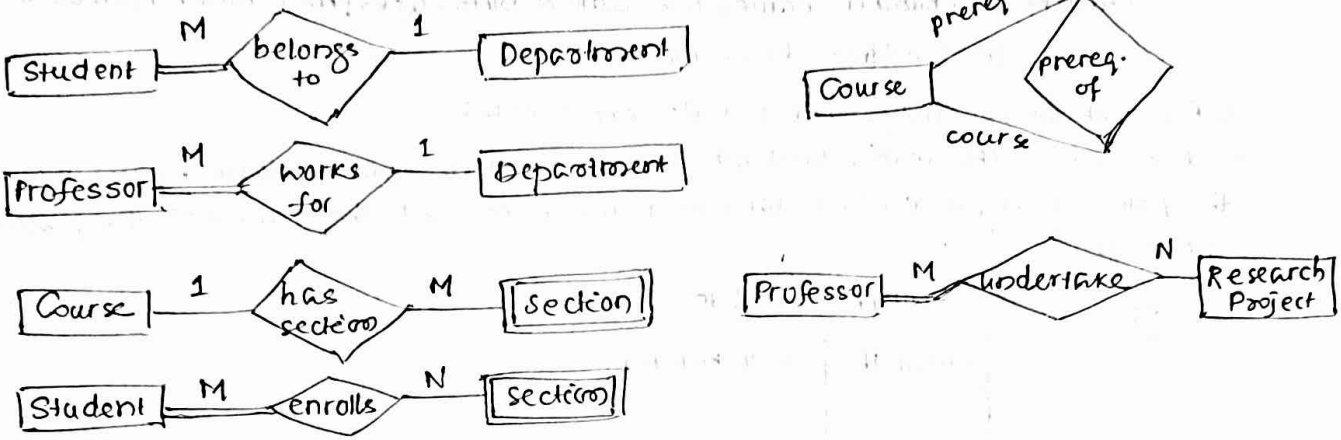
09

Entity Set

- Department
- Student
- Professor
- Course
- Section -- weak entity
- Research Projects



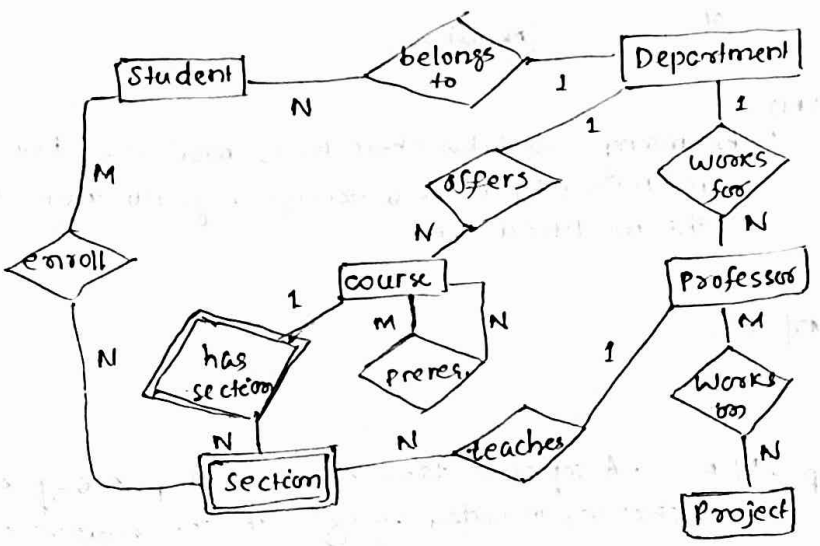
Relationships



L-6 (10.08.2023)(11:00 - 12:00 p.m)

- Student
  - name
  - roll number
  - sex
  - age
  - email id
  - date of birth
  - local address
  - home address
- Professor
  - professor ID
  - name
  - phone number
- Section
  - section ID
  - Timing
  - classRoom
- Department
  - Department ID
  - name
  - location
  - phone number
  - HOD
- Course
  - Course ID
  - name
  - number of credits
- Project
  - Project ID
  - start date
  - end date
  - sponsorer
  - amount

C weak relationship



⊙ While defining the key of a weak entity, we borrow the primary key of the associated strong entity. The combination of this along with the ID of weak entity form discriminator.

→ Ternary relations are splitted into binary relations.

*Designing different tables for the ER Diagram:

(Conversion of ER-Diagram into tables)

- Strong entity set
- Weak entity set
- Relationship

1. Conversion of a strong entity set:

→ Consider all attributes as table column.

Student (roll_number, name, sex, date-of-birth, age, phone-number, email_ID, local_address, home_address)

→ For each strong entity, separate tables are created.

→ For each of the multivalued attributes, create a separate table having the attributes as the primary key of the base table as a reference and the values of the particular attribute.

e.g:

TABLE email_ID

email_ID	roll_number

→ We can break the composite attribute into simple attributes. (capture sub-attributes separately)

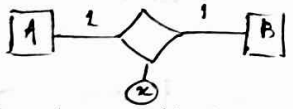
2. Conversion of weak entity:

e.g: Section (course_ID, section_ID, Timing, class_room)

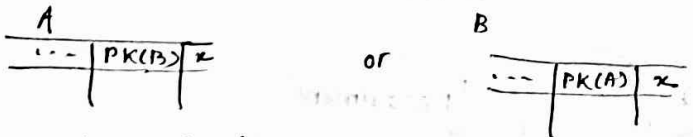
→ Keep all attributes of the entity set along with the primary key of associated strong entity.

3. Relation:

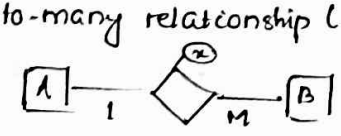
→ One-to-one relationship (1:1)



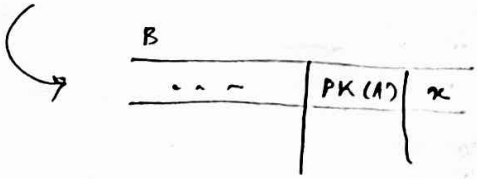
- Include primary key of one entity set as column ; foreign key to the table we want to modify (either A or B)
- If relationship has an attribute, this too is included in the modified table.



→ One-to-many relationship (1:M)



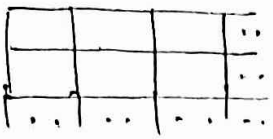
- The many side table (here B) is modified. Primary key of A is referenced by B as a foreign key. Attributes included in the modified table.



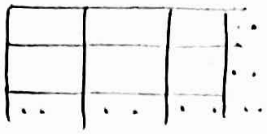
→ Many-to-many relationship (M:N)

- A separate table is created, primary key of both tables are included along with the relationship attributes

A



B



C

PK(A)	PK(B)	x	y

* Creation of Table for the ER-Diagram: (P-09)

• Insertion/ Deletion anomaly.

Student

student_ID /Roll_number	name			sex	date_of_birth	age	local add.		home add.				Dept. ID
	first	mid	Last				h_name	r_no.	h_n	st.	city	PIN	

Emails

Email_ID	Roll_number

Department

Dept_ID	Name	location	Phone_number	HOD	Dept_ID

Professor

Professor_ID	Name	Phone_number

Course

Course_ID	Name	No._of_credits	Dept_ID

Project

Project_ID	start_date	end_date	sponsorer	amount

Section

course ID	section_ID	Timing	Class Room	Professor_ID

Enrollment

Roll_number	Section_ID

Project_Undertaken

Professor_ID	Project_ID

Prerequisite

Course. ID	Course-ID

ex:

F	G
B → CD	B → CDE
AD → E	B → ABC
B → A	AD → E

Ans:

F	G
B^+ = BCDEA	B^+ = BACD = ABCDE
AD^+ = ADE	AD^+ = ADE

'G' is covered by 'F'.

F	G
B^+ = ACDB = ABCDE	B^+ = ABCDE
AD^+ = ADE	AD^+ = ADE

'F' is covered by 'G'.

So, 'F' is equivalent to 'G'.

Q How to identify key of a table from a closure set?

Identify key:

e.g:

A → BC	D → BD
CD → E	ABH → BC
E → C	DH → BC
	D → AEH

NOTE: In L.H.S., for individual attribute, we try to find closure set. (eg. in this case, A, D, E)

A^+ = ABC
D^+ = ABDEH
E^+ = EC

Out of the three, D^+ contains all the attributes. Therefore, we identify 'D' as the key as it contains all the attributes.

Q Find key for given functional dependencies: (Try to find all combinations of attributes)

A → B
BC → E
ED → A

BC^+ = BCE
ED^+ = AED = ABED
AB^+ = AB
BD^+ = BD
AC^+ = ACB = ABCE

combine 3 combinations:

ACD^+ = ACDB = ABCDE (usually we go for ACD+ as single attribute of present)
BCD^+ = BCDE = BCDEA = ABCDE
ECD^+ = ABCDE

Q A → DE
AB → C
B → F
F → GH
D → IJ

A^+ = ADE = ADEIJ
B^+ = BF = BFGH
F^+ = FGH
D^+ = DIJ
AB^+ = ABC = ABCDE = ABCDEF = ABCDEFGH = ABCDEFGHIJ

⇒ AB^+ is the key.

If we add AB^+ with F or D or any other LHS attributes which give all attributes, we derive a set i.e. candidate keys : {AB, ABF, ABD}

Super key (key along with other determinants)

Classification of Functional Dependencies:

(1) Partial F.D.:

→ Usually, non-key attribute depends upon key attributes. But where a particular non-key attribute depends on part of key, it is called a partial F.D.

→ If any of the non-key attributes depends on part of the key, it is called as partial dependencies.

→ e.g.: R = ABCD Key = AB
AB → C
C → D
B → D (B determines D or D depends on B)

B → D is a partial FD as 'D' is a non key & B is part of key AB.

→ Some situations where partial FD is not possible..

↳ e.g.: If key is single attribute or key contains all the attributes or if we have only two attributes present in table.

→ In the following condition, table can't have partial F.D.:

(i) If the primary key consists of a single attribute.
(ii) All the attributes in table are part of the key.
(iii) If the table consists of only two attributes.

(2) Transitive F.D.:

→ If there is a relationship among non-key attributes, then they are called transitive FDs

→ e.g.: R = ABCD
AB → C
C → D (Transitive F.D. as C, D, ∈ non-key set)
B → D
AB is key.

→ In the following condition, table can't have transitive F.D.:

(1) All the attributes in table are part of the key.
(2) If table consists of only two attributes.

(3) Full Functional Dependency:

→ If there is a functional dependency x → y, the removal of an attribute from x makes the F.D. x → y as invalid. Then we say that it is full F.D.

→ e.g.: AB → C

remove B, so this is invalid.

AB → C is a full F.D.

18.05.2023

Ex: F : AB → CD
C → A
D → B

AB^+ = ABCD (key)

Check whether partial F.D. or transitive F.D. is there.

There is no partial F.D.

Transitive F.D. (D → B)

If CD is the key, C → A & D → B are partial F.D.s

If partial F.D. or transitive F.D. are present, it is called anomalies in database.

sol°: split the table (Till when ?)

insertion / updation / deletion anomaly.

Normalization:

It is a process of reducing redundancy from the universal database tables. Therefore, the insertion, deletion and updation anomalies will be removed.

Types of Normal Form:

(1) First Normal form:

If no redundant records exist in the table, then table is said to be in 1NF.

unique records

2. Second Normal Form (2NF)

If a table satisfies two criterias:

(a) it must be in 1NF

(b) There should be no partial dependency

it is said to be in second normal form

Example: $AB \to C$, $A \to DE$, $B \to F$, $F \to GH$, $D \to IJ$

$AB \to C$

$AB^+ \to ABC$ (key) $= ABCDE$

$= ABCDEF$

$= ABCDEFGH$

$= ABCDEFGHIJ$

$A^+ = ADE$

$= ADEIJ$

$B^+ = BF$

$= BFGH$

$A \to DE$, $B \to F$ has partial F.D. (not in 2NF)

* If there is a partial FD, we remove the partial dependent attribute from original table and place it in a separate table along with copy of its determinants.

R1 : ABC

R2 : ADEIJ → A is key (Determinant along with attributes for which we are getting partial F.D.)

R3 : BFGH → B is key

Now we say this table is in 2NF.

Eg: $A \to FC$, $C \to D$, $B \to E$ → partial F.D.

$AB^+ = AFCBE$

$= ABCDEF$ → key

$B^+ = BE$

$A^+ = AFCD$

$R_1 : AB$, $R_2 : AFCD$, $R_3 : BE$ — 2NF

$\Longleftrightarrow$

$R_1 : AB$, $R_2 : AFC$, $R_3 : CD$, $R_4 : BE$ — 3NF

(3) Third Normal Form (3NF)

If a table satisfies 2 criteria, it is said to be in 3NF:

(a) It should be in 2NF

(b) There should be no transitive F.D.

If there is a transitive dependency, we remove this transitive dependent attributes from the 2NF table and place it in a separate table along with copy of determinants.

3NF $R_1 : ABC$

$R_2 : ADEIJ$ (A is key) — 'D' determines IJ (transitive F.D.)

$R_3 : BFGH$ (B is key) — 'F' determines GH (transitive F.D.)

3NF →

$R_1 : ABC$

$R_2 : ADE$

$R_3 : BF$

$R_4 : DIJ$

$R_5 : FGH$ .. in 3NF

(4) Boyce Codd Normal Form (BCNF)

· A table is said to be in BCNF if it is already in 3NF and all the determinants should be the key.

* 3NF can't handle the following situations:
  (i) If the table consists of multiple composite keys.
  (ii) If there is an overlapping attribute among candidate keys.

* Difference between 3NF and BCNF:

3NF	BCNF
(1) It focuses on primary key.	(1) It focuses on candidate keys.
(2) Still has redundancies.	(2) More strict than 3NF
(3) If there is a dependency $x \to y$, it is allowed in 3NF, if either 'x' is a super key or 'y' is the part of key.	(3) If there is a dependency $x \to y$, then it is allowed in BCNF if 'x' is a super key only.
• Lossless decomposition: achieved	.. is hard to achieve.

Example of BCNF:

R(A, B, C, D), F.D.s: $A \to B$, $A \to C$, $C \to D$, $C \to A$.

Candidate keys: A and C

$A^+ = ABCD$

$C^+ = ABCD$

Transaction Processing:

- Assumptions
  - Database server has single processor system.
  - 2-tier DB Architecture
  - Single DB system

- An application program usually contains number of simultaneous transactions.
- In user point of view, transaction is a logical operation. It actually contains a set of micro-operations.

→ While being carried out, transactions satisfy the ACID property:
- Atomicity
- Consistency
- Isolation
- Durability

⊙ Atomicity: A transaction should either be done entirely or not at all performed.
- 'Recovery module' is responsible for enforcing atomicity. (by checking log files).

⊙ Consistency: Application programmer should write efficient program that moves the database from one consistent to another consistent state, after the transaction has been performed.
- 'Application developer' is responsible for enforcing consistency.

⊙ Isolation:
- A transaction starts executing only after any previously executing transaction is completed.
- Isolation is enforced by 'concurrency control module'.
- A particular transaction cannot communicate with another.
- Isolation means separating the transactions.

e.g: A = 500

T1	T2
Read A	Read A
A = A+500	A = A-1000
Write A	
	Write A

(interleaving executions)
↳ DB will move to inconsistent state.

⊙ Durability:
- Enforced by 'Recovery Module'.
- The transactions that are going to be permanently recorded in the database must be successful.
- Log should be maintained only for fail cases and they must not be recorded in the database.

→ 1. (02.09.2023)

Isolation operation:

→ If transactions are sequentially executed, then throughput will be low.
→ Certain regulated interleaf operations are allowed. (Ensured by concurrency control module)
→ In what scenarios database transaction fails?
- ↳ System crash (power failure, network issues)
- ↳ Transaction Program error
- ↳ Transact° aborted by concurrency control manager, due to rule violation.

Partial Transaction Effects: Aborted
Successful Transaction: Recorded

→ All the details recorded in System log.

• System Log:

- Details of the running transactions are recorded in the log.
- Important resource for recovering from crashes.
- Always maintained on reliable secondary storage.

• Log entries:

1. Beginning of transaction
2. Update operation. details - old values - new values
3. Ending transaction — w.r.t. database

→ We are only concerned about read & write operations. Because the rest of the operations are handled by the application program.
(high level program)

• From the database system point of view, only the read and write operation of a transaction are important.

• Other operations happen in memory and do not affect the database on disk.

Notation: (database object)

- R(X) - Transaction reads item 'X'
- W(X) - Transaction writes item 'X'

We can denote database objects as X, Y, Z, etc..

→ Commit: A transaction issues a 'commit' command when (checks)

1. It has successfully completed its sequence of operations.
2. Indicates that its effect can be permanently recorded in the database.

@ If there is an error, it calls the concurrency control module to handle rule violations.

→ Abort: When a transaction issues 'abort.'

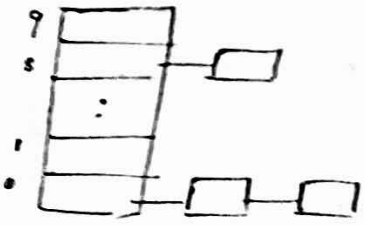
1. indicates that some internal error is there.
2. transaction wants to terminate in the middle.

• System obligations (responsibilities) when abort command has been issued:

→ Ensures that the partial work done by the transaction has no effect on the database on disk.

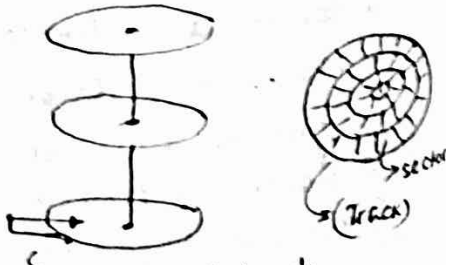
• Database model for transaction processing:

- Granularity of item in database is dependent on the algorithm used. (as block size depends on algorithm)



(Use of Hashing)

Problem:
Collision
↳ solved by probing



2 read/write heads one on each side

Track: divided into no. of blocks

Data mapping: many use hashing algorithm (in file system)

B+ tree: index file structures

→ Transactions operate by exchanging the data with database only. They do not exchange messages between them.

→ DB Model focuses on read/write/commit/abort operations and ignore the in memory operations.

→ Transactions are not nested.

$R_i(X)$ — i - stands for transaction number

↳ read operation / transaction 'i' on database item 'X'.

1. Disk block having 'X' - copied to buffer page if required.
2. Required value(s) is assigned to program variable 'X'.

DA = Salary

DA ↑ memory variable ; Salary ↑ copied to buffer

$W_i(X)$ → write operation / transaction 'i' on database item 'X'.

1. Disk block having 'X' - copied to the buffer page if required.
2. Update the buffer value using program variable 'X'.
3. Transfer the block to the disk. (can do immediately or later) (if further modification is req!

C_i . Commit transaction 'i'

A_i - Abort transaction 'i'

→ 1- (07.09.2023) (11:00-12:00 pm)

. Transaction manager processes no. of transactions at a time to increase throughput and performance. Therefore, in some cases we allow interleaved operations.

. There may arise anomalies, which need to be considered.

· e.g: Write operation on same database by two different simultaneous transactions.

* There are three anomalies:

(1) RW anomaly
(2) WR anomaly
(3) WW anomaly

* Write-Read Anomaly:

T1	T2
1. W(X)	
	2. R(X)
	3. X = X+5;
4. Revert old value of 'X'	
	5. R(X)

Interleaved operation being allowed, T1 & T2 are being executed. Though T2 updates X, as T1 reverts the old value, the second time T2 tries to read will obtain old value of 'X' instead of getting updated value.

↳ Dirty Read operation.

* Read-Write Anomaly:

T1	T2
R(X)	
	R(X)
	W(X)
R(X)	

T1 is reading the database object 'X'. As T2 updates the value, the second time T1 tries to read will get a changed value. (Although T1 has performed no write or update operation)

* Write-Write Anomaly: (Lost update problem)

T1	T2
R(X)	
	R(X)
W(X)	
	W(X)

Whoever will write first, its update will be lost. While both transactions try to update, an update value will be lost.

* Transaction Schedule: Interleaved operations will be allowed in such a way that no anomaly takes place.

→ It is a sequence of interleaved operations of a transaction set.

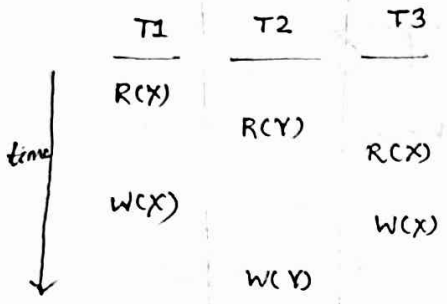


Fig. ref

T is a set of transactions: $T = \{T1, T2, T3\}$

* If we go for serial schedule, then all the operations of T_i will be followed by that of T2 & T3..

Serial schedule: $\underbrace{R1(X), W1(X)}_{T_1}, \underbrace{R2(Y), W2(Y)}_{T_2}, \underbrace{R3(X), W3(X)}_{T_3}$

* We have not allowed any interleaved operation.
* issue: low throughput (reduced performance)
* Advantage: No concurrency issue or problem.

⊙ Serializability: Allow interleaving of operation maintaining some standard.

↳ Serializable schedule
- Interleaving of operations allowed.
- Equivalent to the serial schedules in some sense.

→ The effect of interleaving is same as that of some serial schedules.
i.e., interleaving equivalent to some serial schedules are allowed.

⊙ Conflicting pair of operations:
- operations are involved on the same database item.
- These operations must belong to two different transactions.
- One of the operations must be the write operation.

Example: (Consider Fig. ref):

$R_1(X)$ and $W_3(X)$: involved in same DB item X
: belong to different transactions.
: one is write operation.
∴ $\{R_1(X), W_3(X)\}$

Other two conflicting pairs: $W_1(X)$ and $R_3(X)$ ∴ $\{R_3(X), W_1(X)\}$
$W_1(X)$ and $W_3(X)$ ∴ $\{W_1(X), W_3(X)\}$

- conflict serializable
- view serializable

[* Design of healthcare management system for chronic disease in Odisha.]

Conflict Equivalence:

A schedule 'S1' is said to be conflict equivalent to 'S2' (denoted as $S_1 \cong S_2$) if the relative order of any conflicting pair of operations is same in both 'S1' and 'S2'.

Consider example (fig ref):

S1: $R_1(X), R_2(Y), R_3(X), W_1(X), W_3(X), W_2(Y)$

S2: $R_1(X), R_2(Y), R_3(X), W_2(Y), W_1(X), W_3(X)$

Conflicting pairs:

$\{R_1(X), W_2(X)\}$

$\{R_3(X), W_1(X)\}$

$\{W_1(X), W_3(X)\}$

Relative order of all conflicting pairs of operations in S1 and S2 are the same.

$\Rightarrow$ S1 is conflict-equivalent to scheduler S_2 or $S_1 \cong S_2$.

S3: $R_1(X), R_2(Y), W_1(X), R_3(X), W_3(X), W_2(Y)$

(S1, S3): $R_1(X), W_3(X)$ follow same relative order.

$R_3(X), W_1(X)$ have different relative order. - out of pair violation

Though $W_1(X), W_3(X)$ have same relative order.

↳ $S_1 \not\cong S_3$ (not conflict equivalent)

Conflict Serializable:

A schedule is called conflict serializable if it is conflict equivalent to some serial schedules.

(OR) If we can move non-conflicting operations such that

- The relative order of operations are intact and the schedule becomes a serial schedule.

Example:

S1

T1	T2	T3
R(X)		
	R(Y)	
		R(X)
W(X)		
		W(X)
	W(Y)	

S2

T1	T2	T3
R(X)		
	R(Y)	
W(X)		
		R(X)
		W(X)
	W(Y)	

Conflicting pair of operations:

$\{R_1(X), W_3(X)\}$

$\{W_1(X), W_3(X)\}$

$\{R_3(X), W_1(X)\}$ .... have different relative order

For S2,

$\{R_1(X), W_3(X)\}$

$\{W_1(X), W_3(X)\}$

$\{W_1(X), R_3(X)\}$

↳ These are pairly intact

Therefore S2 is conflict serializable.

- Transactions considered as node.
- Arc: connecting edges transactions.
- Build precedence graph.
- Topological sort
- Result is the serial schedule.

* If the graph is cyclic, non-conflict serializable.
  if acyclic - is conflict serializable.

→ L- (08.09.2023) lecture notes: swayam

View Serializability:

S1 and S2 are two schedules.

→ A schedule S1 is view equivalent to schedule S2 if

(i) T_i reads X in S_1 and also in S_2

(ii) T_i reads X written by T_j both in S_1 and S_2.

(iii) Final write operation done by same transaction both in S_1 and S_2.

example?

→ A schedule S1 is said to be view serializable if it is view equivalent to a serial schedule S2.

Concurrency control using locks:

→ A transaction requests for a lock on database item 'X' for both read and write operations. Then the transaction unlocks the database item 'X' once its job is over.

Locks: binary locks (L / U)

→ At a particular time, on the same database item, a particular transaction will hold the lock

Transactions: {T1, T2} database item: {X}

T1 L(X)

or

T2 L(X)

→ Lock / Unlock situations will be maintained by a 'locking scheduler'.

Locking and Serializability:

T1	T2	T3
L(X) R(X) U(X)		
	L(Y) R(Y) U(Y)	
		L(X) R(X) U(X)
L(X) W(X) U(X)		
		L(X) W(X) U(X)
	L(Y) W(Y) U(Y)	

Lock
Unlock
Read
Write

- Locking alone doesn't guarentee serializability.
- To attain serializability, find conflict equivalent or view equivalent.

Two Phase Locking Protocol: (2 PL Protocol)

(Generate conflict equivalent schedule)

→ All lock requests of transaction preceeds with unlock request.

(OR) A transaction has a locking phase followed by an unlocking phase.

→ If all the transactions follow the 2 PL protocol, the resulting schedule is conflict serializable.

adv → Good for system performance. (Result is good)

disadv → Deadlock may arise .. use deadlock prevention/avoidance mechanism.

→ To detect the deadlock situation a wait-for graph is used. .. detailed in OS.
(directed)

↓ Log Record Entries: (Details of the transaction)

↳ will be stored in reliable secondary storage.

• Who has updated? Timestamp? - Though we are not maintaining these two columns, log records keep them updated.

<start T> • Transaction T has started.

<End T> • Transaction T has completed.

<commit T> • Transaction has successfully completed the work
↳ changes must appear in database-disk.

<Abort T> • Transaction is failed
↳ changes should not be there in database division.

Update records • It is very specific to the logging methods.

Flush log • (Forcefully) Force-write the log entries into the disk.

↓ Undo Logging:

<T, X, V> — V: value (takes old value), X: database object, T: Transaction

→ Transaction 'T' has changed database item 'X' with old value 'V'.

↓ Rules: (Undo logging Rules)

U1: <T, X, V>

If a transaction 'T' modifies the database item 'X', then the log record <T, X, V> must be written to the disk before the new value of 'X' is written to the disk.

U2: If a transaction 'T' commits, then the commit log record must be written to the disk only after all the database items changed by the transaction 'T' have been written to the disk.

```
For each modified database item X
{
    send update entry <T, X, V> to disk
    Write item X to disk
}
Write <commit T> to disk
```

(4) — JAVA solves surprise Quiz

→ L- (09.09.2023) (09:00 - 10:00)

• Transaction T is doing money transfer of Rs. 100/- from account A to account B.
• Consistency requirement: A+B is same before and after the transaction.

m: memory variable
d: database item

Step	Action	m	Mem-A	Mem-B	Disk-A	Disk-B	log
1					500	1500	<start T>
2	Read(A,m)	500	500		500	1500	
3	m := m-100	400	500		500	1500	<T,A,500>
4	Write(A,m)	400	400		500	1500	
5.	Read(B,m)	1500	400	1500	500	1500	
6	m := m+100	1600	400	1500	500	1500	<T,B,1500>
7	Write(B,m)	1600	400	1600	500	1500	
8	Flush log						
9	Output(A)	1600	400	1600	400	1500	
10	Output(B)	1600	400	1600	400	1600	
							<commit, T>
11							
12	Flush log						

Recovery using Undo log:

Examine log and partition transaction into

→ Commited set : Transactions for which <commit T> exists.

→ Incomplete set : Transactions for which <Abort T> exists or <commit T> doesn't exist.

Examine log in reverse direction:

For every update record <T, X, V>

T is committed: do nothing. All is well due to U_2

T is incomplete: Restore value of X on disk as V

- T might have changed some items on disk
- But log entries with old values are on disk due to U_1.

Do <Abort T> log entry for each incomplete T & Flush log.

Undo logging: Crash Recovery:

Crash occurs sometimes before first Flush log.

DB on disk - unchanged.

T - recognized as incomplete.

Log entries of T: if present - used for "undo" <Abort T>

T - resubmitted.

- If crash occurs after 11, <commit T> - on disk : transaction completed
  not on disk : undoing performed.
- If crash occurs after 12, no action required.

Redo logging:

Undo logging: Cancels the effect of incomplete transactions

Ignores the commited transactions

All DB items to be sent to the disk before Txn can commit.

Results in lot of I/O operations, called FORCE-writing.

I/O cannot be buffered.

Redo Logging: - Ignores incomplete Tyns
- Repeats the work of commited Tyns
- Update log entry <T,X,V>: V is the new value.

Buffer utilization might be down

Write-Ahead-Logging Rule

Flag
- Update log records
- commit records
- changed DB items

8 : commit
9 : Flush log.
<T, A, 400>, <T, B, 1600>

Step	Action	Mem-A	Mem-A	Mem-B	Disk-A	Disk-B	log
1							<start T>
2	Read(A,m)	500	500		500	1500	
3	m:=m-100	400	500		500	1500	
4.	Write(A,m)	400	400		500	1500	<T, A, 400>
5.	Read(B,m)	1500	400	1500	500	1500	
6.	m:=m+100	1600	400	1500	500	1500	
7.	Write(B,m)	1600	400	1600	500	1500	<T, B, 1600>
8.							<commit T>
9.	Flush log						
10.	Output(A)	1600	400	1600	400	1500	
11.	Output(B)	1600	400	1600	400	1600	

Recovery using Redo log:

→ Examine logs & partition Tyns into commited set and incomplete set.

→ Examine log in forward direction:

for every update record <T,X,V>

T is Incomplete: Do nothing. DB on disk has no effects.

T is commited: Unsure if all the effects of T are reflected on Disk
- But log entries with new values are on disk (WAL)
- Redo the changes as per the log entry.

Do <Abort T> for each incomplete T and Flush log.

Crash Recovery:

before step-8 : <commit T> not made by T : recognized as incomplete: No action required. DB on disk not changed. (WAL) <Abort T>, resubmit.

after step-8 : <commit T> on disk - T is redone with the help of log entries
not on disk - incomplete - do nothing — WAL - Abort T & resubmit T.

Undo-Redo Logging:

Better flexibility, uses more detailed logging.

<T, X, U, V>
U: old value, V: new value

- Before modifying any DB item X on disk, because of changes made by some txn T, it is necessary that the transaction log <T,X,U,V> must appear on DB.
- <commit T> & disk changes - in any order

8: Flush log
9: output(A)
10: Commit T
11: output(B)

Recovery using Undo-Redo logs:

Redo all committed Txns - order - earliest first
Undo all incomplete Txns - order - latest first
} both necessary

Crash Recovery:

before <commit, T> incomplete T - undone
after <commit, T> committed T - redone

Recoverable Schedule:

Serializability and Recoverability are orthogonal concepts.

↳ no restriction on commit
↳ no restriction on locking

Some schedules have cascading effect

- Cascadeless
- Cascading

Lab: Txn management in SQL
Next: Tuple Relational Algebra

→ L - (15.09.2023) (11:00–12:00)

Relational Algebra:

1) Select (σ)
   Project (π)
   Division (÷)
   Rename (ρ)
   Assignment (←)
   Join (⋈)
   } Basic Relational Algebra functions

* SQL engine converts SQL statements to Relational Algebra form.

2) Union (∪), Intersection (∩), Difference (−) and Cartesian Product (×)
3) Extended relational algebraic functions:
   Minimum, Maximum, Average, count

↳ Select(σ)

↳ Unary operator
↳ syntax: $\sigma_{\text{<cond>}}(\text{Rel})$ —sql→ $\sigma_{\text{<cond>}}(\text{student})$ ← Relation
↳ e.g: Suppose we want to select students of branch computer science & Engineering with age greater than 19.

$$\sigma_{\text{branch} = \text{"CSE"} \text{ and } \text{age} > 19}(\text{student})$$ - will filter the rows only of a relation

→ The conditions are (commutative) allowed to be interchanged (order doesn't affect result)
→ It is commutative in nature.

$$\sigma_{\langle a1\rangle \text{ and } \langle a2\rangle}(\text{Rel}) \equiv \sigma_{\langle a2\rangle \text{ and } \langle a1\rangle}(\text{Rel})$$

→ $\sigma_{\langle c1 \rangle}(\sigma_{\langle c2 \rangle}(R)) \equiv \sigma_{\langle c2 \rangle}(\sigma_{\langle c1 \rangle}(R))$

$\equiv \sigma_{\langle c2 \rangle \text{ and } \langle c1 \rangle}(R)$

↑ conjunction

Composite condition: AND, OR, NOT

→ The comparision operators $<, \leq, >, \geq, =, \neq$ can be used in the condition clause.

Project (π)

↳ Unary operator

↳ Syntax: $\pi_{\langle attributes \rangle}(R)$

↳ Filters the columns

→ e.g: $\pi_{name, age}(Student)$

name	age
⋮	⋮

all rows

→ Commutative is not allowed in project statement.
If allowed some properties of table will be lost.

SQL

```
SELECT name,age
FROM students
WHERE branch = "CSE"
  AND age > 19;
```

↓ convert into Relational Algebra using select & project

$\pi_{name, age}(\sigma_{branch = "CSE" \text{ and } age > 19}(students))$

Nested query should show the desired output.

- If all of the operations are performed on the top of base table, then order won't matter usually.

⊙ Go in such a way that inner query should produce sufficient data for outer query.

Division (÷)

↳ binary operator

$R = ABC$

$S = AB$

$R \div S = C$

↳ Subset of $R - S$

e.g:

R

A	B	C
a_1	b_1	c_1
a_2	b_1	c_2
a_2	b_2	c_1
a_1	b_2	c_2
a_2	b_1	c_2
a_1	b_2	c_3
a_1	b_2	c_4
a_1	b_1	c_5

S

C
c_1

R÷S

A	B
a_1	b_1
a_2	b_2

e.g.2

S

A	B
a_1	b_1
a_2	b_2

R

A	B	C	D
a_1	b_1	c_1	d_1
a_2	b_2	c_1	d_1
a_1	b_1	c_2	d_2
a_1	b_1	c_3	d_3
a_2	b_2	c_3	d_3

R÷S

C	D
c_1	d_1
c_1	d_1
c_2	d_2
c_3	d_3
c_3	d_3

duplicates must not be kept

Now:

C	D
c_1	d_1
c_2	d_2
c_3	d_3

e.g.3 ...

* Union, Intersection, Difference:

R

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

S

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

R ∪ S

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

R ∩ S

A	B	C
a_1	b_1	c_1
a_1	b_2	c_3

R − S

A	B	C
a_2	b_1	c_2

S − R

A	B	C
a_1	b_1	c_2
a_3	b_2	c_3

* Intersection (∩) can be expressed using union and difference:

$$R \cap S = (R \cup S) - ((R - S) \cup (S - R))$$

* Join (⋈) can be expressed using select and cartesian product:

$$R \bowtie_c S = \sigma_{<c>} (R \times S)$$

* Division (÷) can be expressed using project, product and difference:

$R \div S = T$

$$T_1 \leftarrow \pi_{<\text{attr}>} R$$

$$T_2 \leftarrow \pi_{<\text{attr}>} ((S \times T_1) - R)$$

$$T = T_1 - T_2$$

* Basic operators are:

a. Select (σ)
b. Project (π)
c. Union (∪)
d. Difference (−)
e. Cross Product (×)

Next class:
Query optimization

Q1

T_1	T_2
R(A)	
W(A)	
	R(B)
	W(B)
R(B)	
W(B)	
	R(A)
	W(A)

Schedule S is:
(i) view serializable but not conflict serializable
(ii) conflict serializable but not view serializable
(iii) Both view & conflict serializable
(iv) Neither view nor conflict serializable

Q2 Options same as those of Q1)

T1	T2
R(Q)	
	W(Q)
W(Q)	

Read Transact° management
↳ Test tomorrow