Database Systems

- Database Management System (DBMS) contains information about a particular enterprise (interrelated data), that is both convenient and efficient to use.

Ex:
- Universities (new students, instructors, courses, assign GPA's to students, etc.)

— Disadvantages of file systems

- Multiple file formats, duplication of different files.
- Hard to add new constraints or change existing ones.
- Difficult to access data w/o writing new program to carry out that task.
- Uncontrolled concurrent accesses (by multiple users) leading to inconsistencies
- Security problems.

— Levels of Abstraction

- Physical level (describes how a record is stored)
- Logical level (describes data stored in database/record and relationships among data)
- View level (application programs hide details of datatypes, can also hide information for security purposes)

→
- Logical Schema (overall logical structure of database) (analogous to type of information of variable)
- Physical Schema (overall physical structure of database)

- Instance (actual content of database at a particular point

in time) (analogy is value of variable).

☆ Data Models

- A collection of tools for describing data, data relationships, data semantics, data constraints.

— Relational Model

- All the data is stored in various tables

Ex-

(INSTRUCTOR TABLE)

ID	name	dept_name	salary
22222	Einstein	Physics	95000
12345	Wu	Finance	75000
32154	Gold	Music	120000
67891	Singh	History	87000
⋮	⋮	⋮	⋮

(DEPT. TABLE)

dept_name	building	budget
Comp Sci.	Taylor	100000
Biology	Watson	90000
Music	Taylor	220000
Finance	Packard	57000
History	Packard	70000

- Data Definition Language (DDL) is a specific notation used for defining the database schema.

Ex-

```
create table instructor (
        ID          char(5),
        name        varchar(20),
        dept_name   varchar(20),
        salary      numeric(8,2));
```

- DDL compiler ~~stores~~ generates a set of table templates stored in data dictionary which contains metadata about the database schema, primary key (ID uniquely identifies instructors) and authorization.

- Data Manipulation Language (DML) is the language for accessing and manipulating data organized by appropriate data model. (aka query language) (Ex:- SQL)

- Extensible Markup Language (XML), originally intended as document markup language and not a database language, provides a way to specify new tags and create nested tag structures.

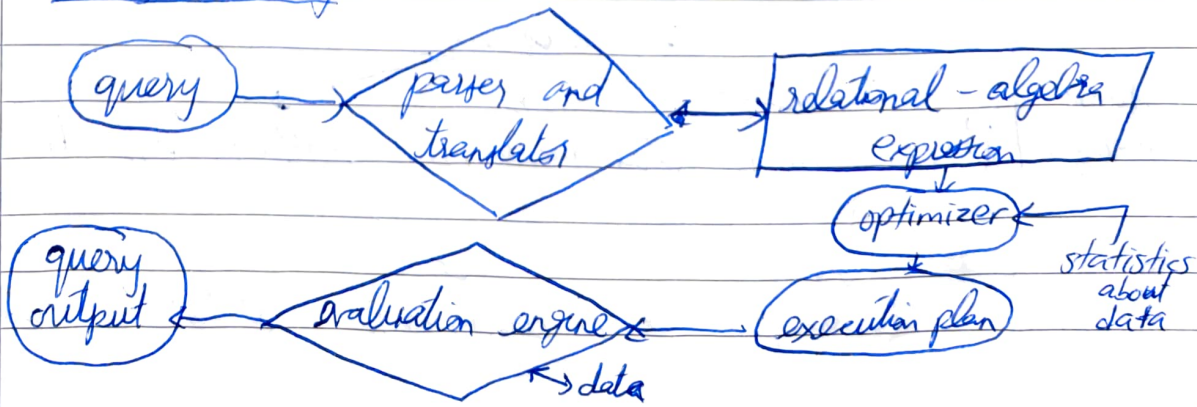
☆ Database Engine

— Storage Management

- Program module that provides the interface between low-level data stored in database and application programs and queries submitted to the system.

- It is responsible for interaction with OS file manager and efficient storing, retrieving and updating of data.

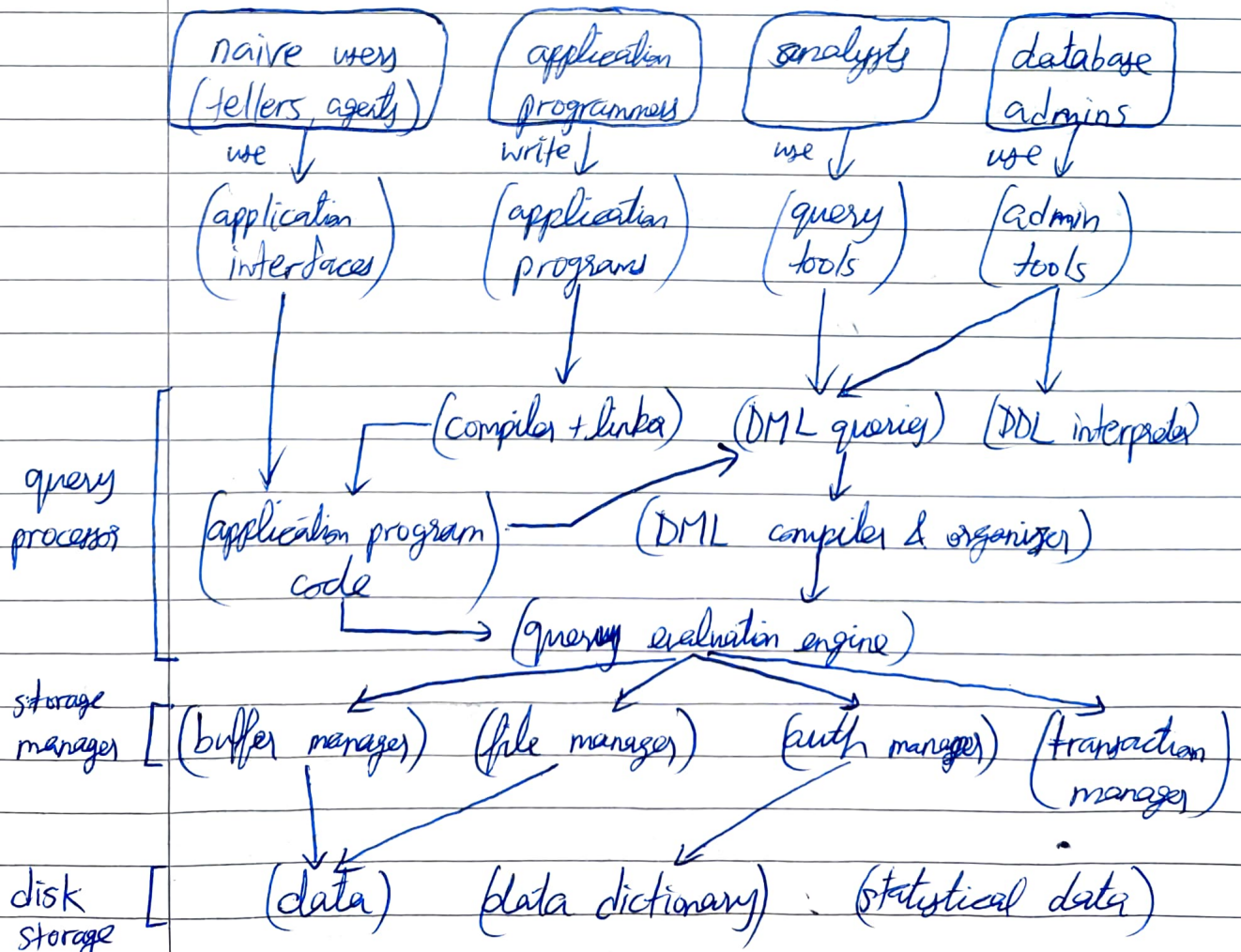
— Query Processing



— Transaction Management

- Transaction is a collection of operations that perform a single logical function in a database application.
- Transaction-management component ensures that database remains correct/consistent state during system/power failures.
- Concurrency-control manager controls interaction among concurrent transactions ensuring consistency.

Note: Architecture of database systems is greatly influenced by underlying computer system on which database is running; centralized, client-server, parallel or distributed.



RELATIONAL MODEL

- In a relational model (table), each column is an attribute and each row represents a tuple.
- Set of allowed values for each attribute is called domain of attribute, which are required to be atomic/indivisible.

Ex: instructors = (ID, name, dept_name, salary)
↳ relation schema — attributes

- Superkey: Let $K \subseteq R$, K is a superkey if values of K are sufficient to identify unique tuples. Ex:- {ID}, {ID, name}
- Candidate Key: Minimal (smallest) superkey of R. {ID} (w/o redundancy)
- Primary Key: First key used to identify only one instance of an entity.
- Foreign Key: Value in one relation must appear in another referencing / referenced relation.

★ Relational Algebra Operators

- Selection (σ): Returns rows of input relation that satisfy the predicate.
  Ex:- $\sigma_{salary>=85000}(instructors)$
- Projection (Π): Returns specified attributes/columns of input relations and removes duplicate tuples.
  Ex:- $\Pi_{ID, salary}(instructors)$

- **Cartesian Product (×)**: Output pairs of rows from two input relations having same value on all attributes having same name.
  Ex:- instructs × department

- **Union (∪)**: Returns union of tuples from two input relations.
  Ex:- $\Pi_{name}(instructs) \cup \Pi_{name}(student)$
  (tuples in instructs + tuples in student)

- **Set Difference (−)**: Returns set difference of tuples from two input relations.
  Ex:- $\Pi_{name}(instructs) - \Pi_{name}(student)$
  (tuples in instructs − tuples in student)

- **Natural Join (⋈)**: Returns pairs of rows from two input relations where attributes are of same name and same value.

Ex

(r)

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

(s)

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	ε

$r \bowtie s \Rightarrow$

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

$$\Pi_{A, r.B, C, r.D, E}\left(\sigma_{r.B = s.B \wedge r.D = s.D}(r \times s)\right)$$

- Intersection ($\cap$): Returns common tuples from two input relations.

  Ex:- $r \cap s = r - (r - s)$

- Renaming a Table: For a relation r, $\rho_s(r)$ would rename the table to s.

INTRODUCTION TO SQL

- Domain Types

  - char(n) (Fixed length character string, with user-specified length n)
  - varchar(n) (Variable length character string, with user-specified max length n)
  - int (integer, machine dependent)
  - numeric(p,d) (fixed point number, with precision p digits and d decimal digits)
  - float(n) (precision of atleast n digits)

- Create Table Construct

```
create table instructor (
        ID          char(5),
        name        varchar(20) not null
        dept_name   varchar(20),
        salary      numeric(8,2));
        primary key (ID),
        foreign key (dept_name) references department);
```

__/__/__

☆ Updates to tables

- Insert: insert into instructs values ('10211', 'Smith', 'Biology', 66000)
- Delete: remove all tuples.
  → delete from student
- Drop: drop table r
- Alter: alter table r add gender varchar(1)
  (for the new attribute, all tuples are assigned with null values.)

  alter table r drop gender

☆ Basic Query Structure (select ... from ... where ...)

— select

- list the attributes desired in the result of query (more like projection operation (Π))

  select name from instructs.

- To remove duplicates
  select distinct dept_name from instructs
  Otherwise,
  select all dept_name from instructs.
- To select all attributes / entire table

  select * from instructs.

__/__/____

- select '437' → table with 1 column and 1 row with value 437.
  select 'A' from instructor → table with 1 column and n rows, each having value 'A'.
- You can rename columns using 'as' clause:

```
select ID, name, salary/12 as monthly_salary;
```

– **where**

- Specifies conditions that result must satisfy.

```
select name from instructor where
    dept_name = 'Comp Sci' and salary > 80000
                (or, not)
```

– **from**

- Lists the relations involved in the query.

→ select * from instructor, teaches (Cartesian product)
(instructor x teaches)

Ex: Find names of all instructors in Art department who have taught some course and course_id.

```
select name, course_id
From instructor, teaches
where instructor_id = teaches_id and instructor.dept_name =
                                    'Art';
```

– **String Operations**

- To check for matching substrings, operator 'like' is used in the where clause.

__/__/____

- % character matches any substring
  _ character matches any character.

Ex: select name
from instructor
where name like '%dar%';
Sundar
(name that includes substring 'dar')

- Escape character (\)

- 'Intro%' → string beginning with 'Intro'
  '___' → matches string of exactly 3 characters
  '___%' → matches string of atleast 3 characters.

- Concatenation (||)

– Ordering tuples

select distinct name
from instructor
order by name desc/asc (descending/ascending)
(alphabetically)

– where (Contd.)

select name
from instructor
where salary between 90000 and 100000

select name, course_id
from instructor, teaches
where (instructor.ID, dept_name) = (teaches.ID, 'Biology');
(tuple comparison)

Note: Set operations such as union, intersect or except can be used in between ~~where clause~~ select clauses.

- To retain duplicates, use 'all'
  Suppose a tuple occurs m times in r and n times in s then,
  $(m+n)$ times in r union all s
  $\min(m,n)$ times in r intersect all s
  $\max(0, m-n)$ times in r except all s

- null signifies an unknown value that doesn't exist.

Ex: select name from instructor where salary is null.

- Three-valued logic:
  OR: (null or true) = true
  (null or false) = null
  (null or null) = null
  AND: (null and true) = null
  (null and false) = false
  (null and null) = null
  NOT: (not null) = null

— Aggregate Functions

① avg: average value ④ sum: sum of values
② min: minimum value ⑤ count: no. of values.
③ max: maximum value

Ex: No. of tuples in 'course' relation.

select count(*) from course

__/__/____

⑥ group by : select dept_name, avg(salary) as avg_salary
from instructs
group by dept_name;

⑦ having : It is like where clause for 'group by' clause
→ select dept_name, avg(salary)
from instructs
group by dept_name
having avg(salary) >42000;

Note: All aggregate operations except count(*) ignore tuples with null values on attributes.

★ Nested Subqueries :

• Subquery is a select-from-where expression nested within another query.

– Subqueries in 'where' clause

select distinct course_id
from section
where semester = 'Fall' and year = 2009 and
course_id in (select course_id
from section
where semester = 'Spring' and year = 2010);
(Courses offered in Fall 2009 and Spring 2010)

Ex: Find total no. of students who have taken course taught by instructor with ID 10101
(set membership)

```
select count (distinct ID)
from takes
where (course_id, sec_id, semester, year) in
        (select course_id, sec_id, semester, year
         from teaches
         where teaches.ID = 10101 );
```

Ex: (Ex: comparison) find names of instructors with salary greater than atleast one / all from Biology dept.

(Classic)
```
select distinct T.name
from instructor as T, instructor as S
where T.salary > S.salary and S.dept_name = 'Biology';
```

```
select name
from instructor
where salary > some/all (select salary
        (ANY)            from instructor
                         where dept_name = 'Biology');
```

Ex: (Empty Relation)
```
select course_id
from section as S
where semester = 'Fall' and year = 2009 and
      exists (select *
              from section as T
              where semester = 'Spring' and year = 2010 and
                    S.course_id = T.course_id );
```

Note: 'unique' clause tests whether subquery has any duplicate tuples in it's result.

__/__/__

— Subqueries in 'from' clause

Ex: Find avg instructors salaries to those depts where average salary is greater than $42000.

```
select dept_name, avg_salary
from (select dept_name, avg(salary) as avg_salary
      from instructs
      group by dept_name)
where avg_salary > 42000;
```

• 'with' clause provides a way to define a temporal relation. (table)

Ex: ~~with max-budget~~

```
with dept_total (dept_name, value) as
     (select dept_name, sum(salary)
      from instructs
      group by dept_name),
dept_total_avg (value) as
     (select avg (value)
      from dept_total)
select dept_name
from dept_total, dept_total_avg
where dept_total.value > dept_total_avg.value;
```

— Subqueries in 'select' clause

```
select dept_name,
     (select count (*)
      from instructs
      where department.dept_name = instructs.dept_name)
as num_instructors
```

from department;

✱ Modification of DB

– Deletion

delete from 'instructs where dept_name = 'Finance';

– Insertion

insert into course(course_id, title, dept_name, credits)
values ('CS-437', 'DBMS', 'Comp Sci', 4);

– Updates

(i) update instructs
set salary = salary * 1.03
where salary > 100000;
update instructs
set salary = salary * 1.05
where salary <= 100000;

(ii) update instructs
set salary = case
when salary < 100000 then salary * 1.05
else salary * 1.03
end

Intermediate SQL

JOIN Operations

- It is a Cartesian product which requires that tuples in both relations match (under some condition). Used as subquery in from clause.

(i) Left Outer Join (all tuples in left relation + attributes)
(if ~~does~~ not value on right, null)
(ii) Right Outer Join (all tuples in right relation + attributes)
(if no value on left, null)
(iii) Full Outer Join (Left + Right Outer Join)
(iv) Inner Join (strict on condition) (more like intersection)

Views

- A virtual relation that hides certain data from view of certain users.

Ex:-
```
create view faculty as
    select ID, name, dept_name   (w/o salary)
    from instructor
selec name
from faculty
where dept_name = 'Biology'
```

— Expanded View

```
create view physics_fall_2009_watson as
(select course_id, room_number
 from (select course.course_id, building, room number
```

```
from course, section
where course.course_id = section.course_id
    and course.dept_name = 'Physics'
    and section.semester = 'Fall'
    and section.year = '2009')
where building = 'Watson');
```

- View relation V_1 is said to be directly dependent on view relation V_2 if V_2 is used in expression defining V_1.
- View relation V_3 is said to be recursive if it depends on itself.
- Views can be updated using insert clause. However SQL prefers updates on simple views where (a) FROM clause has only one base table (b) ~~SELECT clause uses~~ Uses only attribute names from table in SELECT clause (c) missing attributes set to NULL (d) no group by/having.

Ex

```
create view instructs_budget as
select instructs.ID, instructs.Name, instructs.dept_name,
department.Budget
from instructs JOIN department
ON instructs.dept_name = department.dept_name.

update instructs_budget set Budget = 700000 where
    dept_name = 'History';          (Error)
```

(Since view is derived from multiple tables, database does not know whether to update one row/multiple rows)
(What if multiple instructors belong to History dept)

__/__/____

Ex: create view history_instructors as
select *
from instructs
where dept_name = 'History';

insert into history_instructors values ('25566', 'Brown', 'Biology', 100000) (Error)
(It will give error, since it violates the condition that only allows rows where dept_name = History)
(Even though base table does not have any restriction)

— Materialized View

- Database object that stores result of a query in a table as a physical table for faster access (saved)
- Since data can become outdated when original table changes, this view can be manually, periodically or automatically refreshed.

Ex: CREATE MATERIALIZED VIEW sales_summary AS
SELECT region, SUM(sales) AS total_sales
FROM sales GROUP BY region;

REFRESH MATERIALIZED VIEW sales_summary;

☆ Transactions

- Unit of work in a database that ensures that a series of operations are fully completed or rolled back if something goes wrong (atomic transaction)

__/__/____

- In most databases, transactions start automatically.
- COMMIT (saves changes permanently)
  ROLLBACK (cancels transaction, undoing all changes)

✱ Integrity Constraints

- Guard against accidental damage to database, ensuring that changes do not result in loss of data consistency.

Exi- Customer must have non-null phone number.

(a) not null
(b) unique (states that attributes form candidate key)
(permitted to be null unlike primary key)
(c) primary key
(d) check (P) where P is predicate
(check (semester in ('Fall', 'Winter', 'Spring', 'Summer'))
(e) foreign key (referential integrity)

Exer (Violation)

```
CREATE TABLE person (
        ID char(10),
        name char(40),
        mother char(10),
        father char(10),
        primary key ID
        foreign key father references person,
        foreign key mother references person);
```

To insert a tuple, either insert father and mother before inserting person, or set them to null initially.

__/__/__

☆ Built-in Data types in SQL

(a) date '2005-03-17'
(b) time '9:00:30'
(c) timestamp '2005-03-17 9:00:30.75'
(d) interval '2 hours' (usually +/- with time/date)

☆ User-defined Datatypes

```
CREATE TYPE Dollars AS NUMERIC(12,2) FINAL;
(Use)
CREATE TABLE DEPARTMENT (
        .....
        budget Dollars);
```

– Domains (restricts allowed values) (even not null)

```
CREATE DOMAIN degree_level Varchar(10)
CONSTRAINT degree_level_test
check (value in ('Bachelors', 'Masters', 'Doctorate'));
(Use)
CREATE TABLE students (
        .....
        degree degree_level);
```

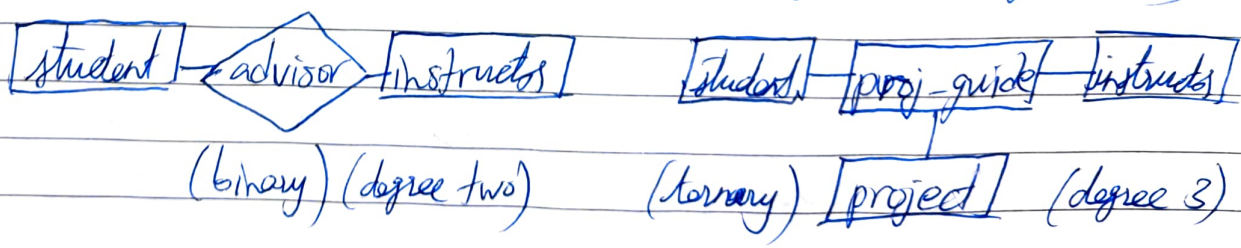
Note ∧ (and), ∨ (or), ¬ (not) in propositional calculus.

—x—

Entity - Relationship (ER) Model

- Models an enterprise as a collection of entities & relationships
- Entity is a thing / object in the enterprise that is distinguishable from other objects. Described by set of attributes
- Relationship is an association among several entities. Represented by ER diagram.
- ER data model was developed to facilitate database design by allowing specification of enterprise schema representing overall logical structure of database.
- Entity set is set of all entities of same type sharing same property.

  Eg: course = (course_id, title, credits)
- Relationship set is a mathematical relation among $n \geq 2$ entities, each taken from entity sets.
- Binary relationship involves two entity sets (degree two)



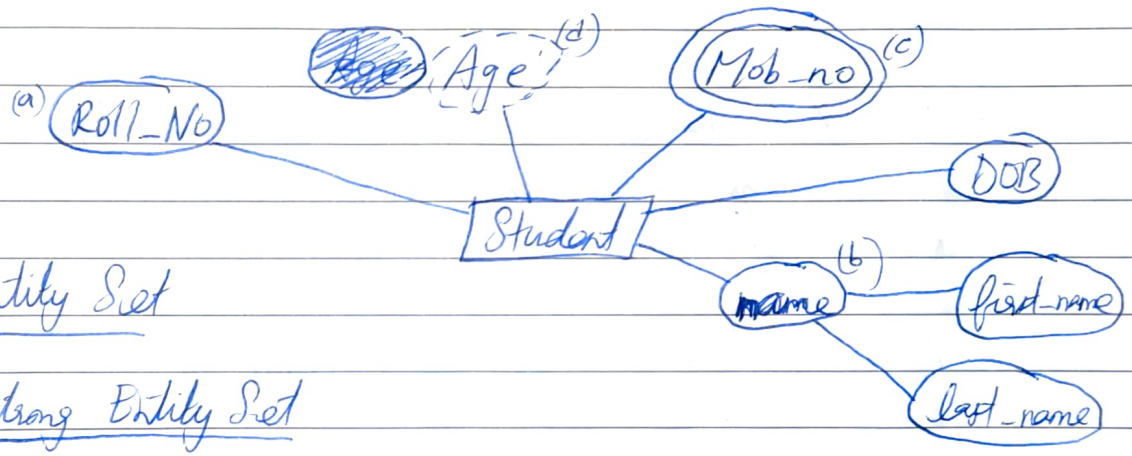
- For binary relationship set, mapping cardinality must be of:
  (a) one - to - one
  (b) one - to - many
  (c) many - to - one
  (d) many - to - many

__/__/__

✱ Complex Attributes

(a) Simple attributes (can't be classified further)
(b) Composite attributes : name (first_name, middle_name)
(c) Single-valued and multi-valued (multiple phone numbers)
(d) Derived attributes (age from date_of_birth)

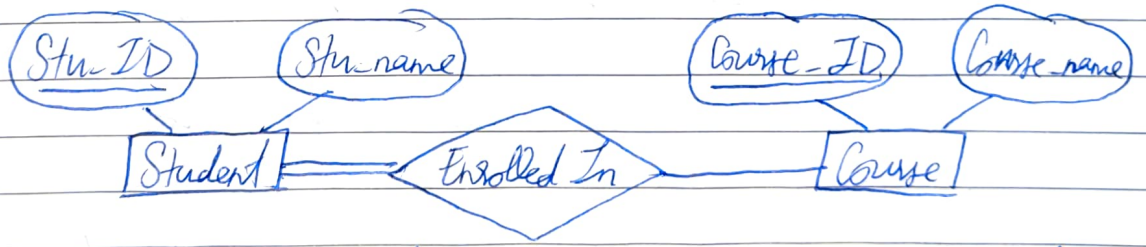
- Domain is set of permitted values for each attribute.



✱ Entity Set

— Strong Entity Set

- Entity set that contains sufficient attributes to uniquely identify all its entities (primary key exists)

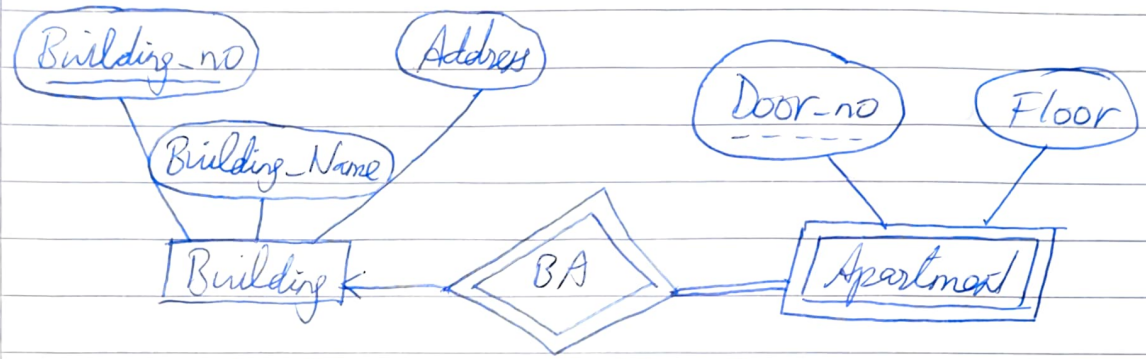


- Here two strong entities Student and Course are related to each other. Here, primary keys are underlined.
- Double line between entities signifies total participation (every student must enrol in atleast one course)
- Single line means partial participation (there might be some courses for which no enrolments are made)

__/__/____

- Weak Entity Set

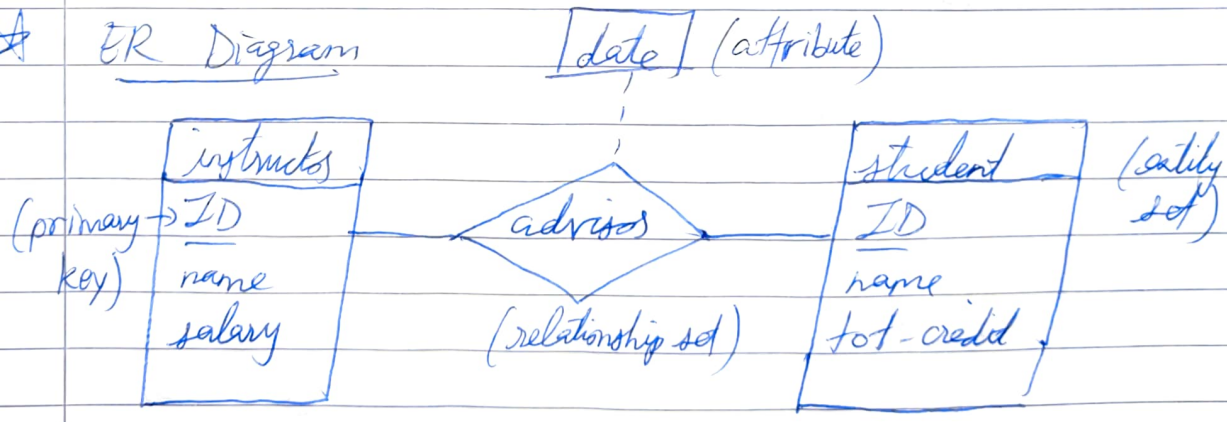
- Entity set does not contain sufficient attributes to uniquely identify entities. No primary key. However it contains a partial key called as discriminator that can identify a group of entities from entity set.



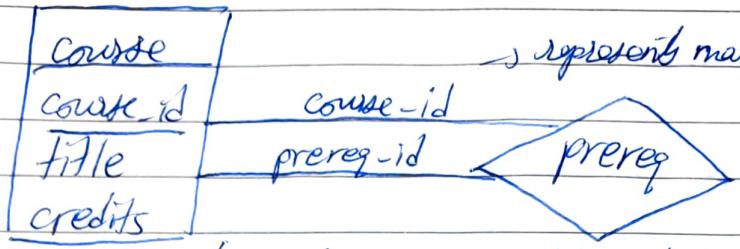
- Here Apartment is weak entity set. Door_no is discriminator which cannot uniquely identify an apartment alone since there can be many other buildings with same door number.

- A weak entity set is one whose existence is dependent on another entity called identifying entity, and discriminator.

★ ER Diagram



- Roles (function/part an entity plays in a relationship when same entity set appears more than once)



→ represents main course

(both from same entity set but play diff roles)

- Cardinality Constraints (→ one, —— many)

(i) One-to-one relationship

A student is associated with at most one instructs via advisor and vice versa

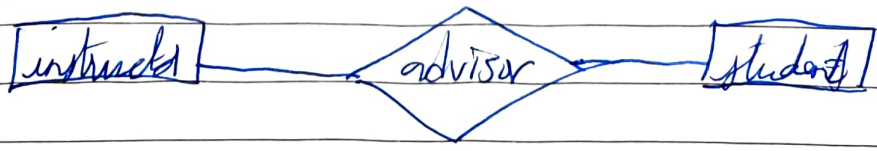
(ii) Many-to-one relationship

Instructs is associated with atmost one student
Student is associated with several instructors

(iii) One-to many relationship

Instructs is associated with several students
Student is associated with atmost one student.

(iv) Many-to many relationship.



__/__/____

Instructs is associated with several students and vice versa.

- Total participation implies that every entity in entity set participates in atleast one relationship.

(min .. max)

Here min value of 1 → total participation
max value of 1 → entity participates in almost one relationship.
max value of * → no limit.

Note Primary key for weak entity set = (primary key of related strong entity set + discriminator)

- Example of entity with Complex Attributes:

instructor
ID
name
first_name
middle_initial
last_name
address
street
street_number
street_name
apt_number
city
state
zip
{phone number}
date_of_birth
age()

- To represent strong entity set,
  student (ID, name, tot_cred)
  To represent weak entity set,
  section (course_id, sec_id, sem, year)
  ↳ from identifying strong entity set

- To represent relationship set (advisor)
  advisor = (student_id, instructs_id)

- For entity sets with composite attributes, either prefix is omitted (name_first_name) or all component attributes are listed out. (w/o multivalued attributes)

instructs (ID, first_name, middle_name, last_name, street_number, street_name, apt_number, city, state, zip_code, date_of_birth)

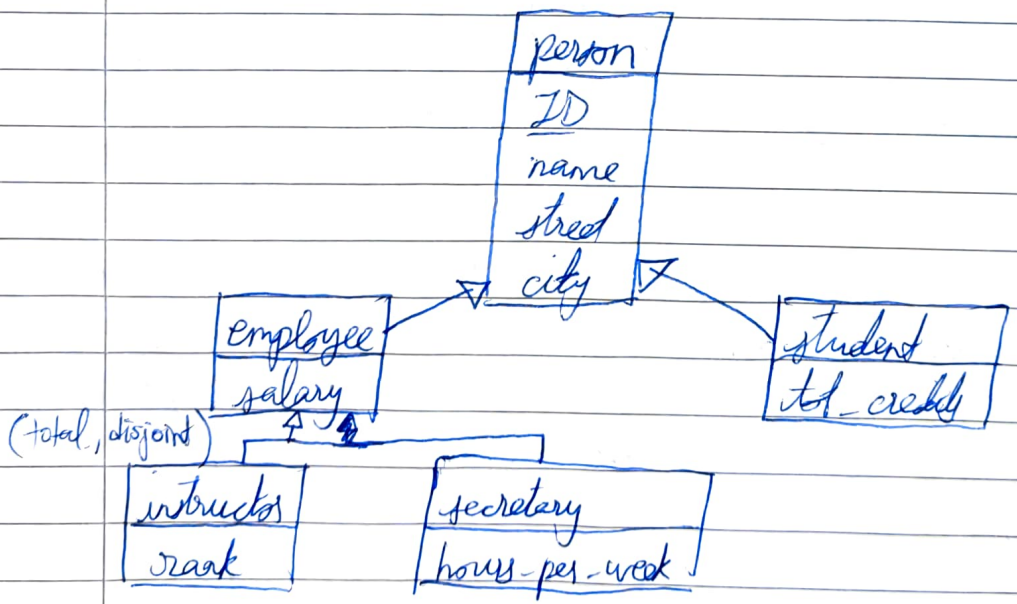
- For multivalued attribute, we use a seperate schema having attributes corresponding to primary key of entity set and that of multivalued attribute.

instructor_phone = (ID, phone_number)
= (22222, 9876543210)
= (22222, 1234567890)

☆ Specialization

- It is a Top-Down design approach in ER modelling. It involves dividing an entity set into subsets (low level entity sets) having distinct attributes or participate in unique relationships.

- Attribute inheritance means that low level entities inherit all attributes and relationships of high-level entity.



- Overlapping : A person can be both employee and student at the same time
- Disjoint : An employee can either be instructor or secretary not both.

- To represent specialization via schemas,

(i) For schema for each low entity set, include primary key of higher level entity set and local attributes.

person (ID, name, city, street)
student (ID, tot_cred)
employee (ID, salary)

(Cons: Getting info about an employee requires accessing both higher & lower level schema)

(ii) Form a schema for each entity set with all local and inherited attributes

person (ID, name, street, city)
student (ID, name, street, city, tot_cred)
employee (ID, name, street, city, salary)

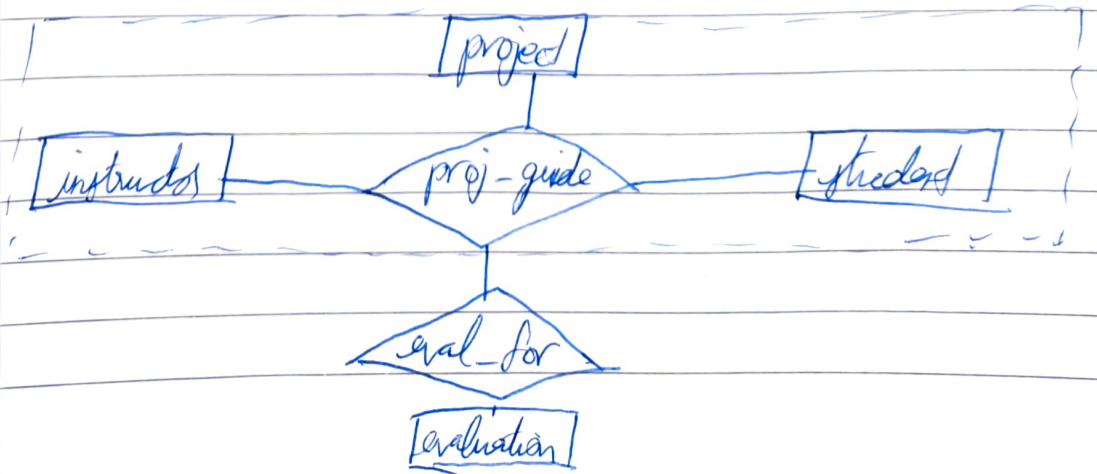
(For people being both student and employee, storing name, street and city would be redundant)

☆ Generalization

- Opposite of Specialization, it is a Bottom-Up design process combining entity sets sharing same features into higher level entity set.

☆ Aggregation

- Given the ternary relationship proj-guide, suppose we want to record evaluations of student by guide on a project

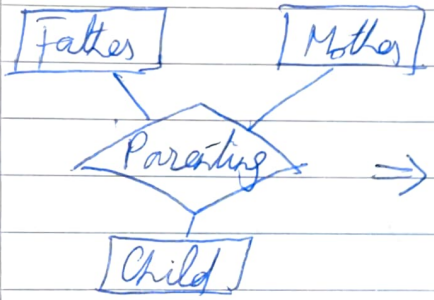


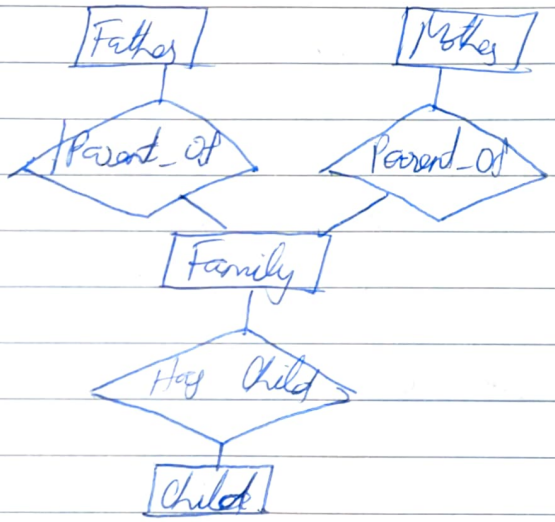
__/__/__

- Instead of connecting project, instructs and student to eval_for causing redundancy, we treat the ternary relationship as an abstract entity. thus allowing relationships between relationships.

eval_for (s_ID, project_id, i_ID, evaluation_ID)

Note Some non-binary relationships may be better represented using binary relationships.





RELATIONAL DATABASE DESIGN

- Functional dependency (FD) is a rule of the form $X \rightarrow Y$ where values of attribute set X can uniquely determine values of attribute set Y

- To avoid redundancy, relations are decomposed into smaller relations / tables.

(i) **Lossy Decomposition**: Occurs when relation is divided into smaller sub-relations, and original relation cannot be reconstructed without introducing extra / incorrect tuples.

A	B	C
1	x	p
2	x	q
3	y	p

A	B
1	x
2	x
3	y

$R_1(A,B)$

B	C
x	p
x	q
y	p

$R_2(B,C)$

$R_1 \bowtie R_2$

A	B	C
1	x	p
1	x	q
2	x	p
2	x	q
3	y	p

→ extra (false positives)

(ii) **Lossless-Join Decomposition**: Original relation is reconstructed

R

A	B	C
α	1	A
β	2	B

$(R_1 \bowtie R_2 = R)$

A	B
α	1
β	2

R_1

B	C
1	A
2	B

R_2

First Normal Form

- Domain is atomic if elements are indivisible.

→ • R is in 1NF if domains of all attributes of R are atomic.

Note: Set of names are composite attributes (non-atomic) these complicate storage and lead to repeated / redundant storage of data.

Ex: Consider r(A, B)

A	B
1	4
1	5
3	7

Here, A → B does not hold (A is not unique) but B → A does hold.

- K is ~~super~~ superkey for relational schema R if K → R.

- K is candidate key iff K → R and for no α ⊂ K α → R.

- A functional dependency is trivial if β ⊆ α (α → β)

BCNF (Boyce-Codd Normal Form)

- α → β is trivial (β ⊆ α) (or)
- α is superkey for R

- If violated, we can decompose R into
  $R_1 (\alpha \cup \beta)$
  $R_2 (R - (\beta - \alpha))$

__/__/____

Ex:- instructor_dept (ID, name, salary, dept_name, building, budget)
(α) (β)

F : dept_name → building, budget

Not in BCNF since dept_name is not superkey in R.

Decompose R into:

$\alpha \cup \beta$ – R_1 (dept_name, building, budget)

$(R - (\beta - \alpha))$ – R_2 (ID, name, salary, dept_name)

Note • A decomposition is called dependency preserving, if we can check all functional dependencies by looking at individual tables, w/o joining them. (if all FD's are available in respective sub-relations)

– Third Normal Form

- $\alpha \to \beta$ is trivial $(\beta \subseteq \alpha)$ (or)
- α is superkey for R (or)
- Each attribute A in $(\beta - \alpha)$ contained in candidate key for R.

– Closure of Set of FD's (F^+) (using Armstrong's axioms)

- Set of all FD's logically implied by F is closure of F.

- If $\beta \subseteq \alpha$, then $\alpha \to \beta$ (reflexivity)
  If $\alpha \to \beta$, then $\gamma\alpha \to \gamma\beta$ (augmentation)
  If $\alpha \to \beta$ and $\beta \to \gamma$, then $\alpha \to \gamma$ (transitivity)

- $\alpha \to \beta$, $\alpha \to \gamma$, then $\alpha \to \beta\gamma$ (union)
  $\alpha \to \beta\gamma \Rightarrow \alpha \to \beta$ and $\alpha \to \gamma$ (decomposition)
  $\alpha \to \beta$, $\gamma\beta \to \delta$, then $\alpha\gamma \to \delta$ (pseudo-transitivity)

__/__/__

Ex: $R = (A, B, C, G, H, I)$
$F = \{A \to B,\ A \to C,\ CG \to H,\ CG \to I,\ B \to H\}$

Some members of F^+:
$A \to H$ $(A \to B,\ B \to H)$
$AG \to I$ $(AG \to CG$ (augment), $CG \to I)$
$CG \to HI$ $(CG \to H,\ CG \to I)$

$(AG)^+ = \{AG\}$
$= \{ABCG\}$ $(A \to C,\ A \to B)$
$= \{ABCGHI\}$ $(CG \to H,\ CG \to I\}$

$\therefore$ AG is superkey
For AG to be candidate key, AG should be minimal
$(A)^+ = \{A\} = \{ABC\} = \{A, B, C, H\}$ ✗
$(G)^+ = \{G\} = \{G\}$ ✗
$\therefore$ Since AG is minimal, it is candidate key.

Note: From attribute closure, we can

(i) Test for superkey (Compute α^+, check if α^+ contains all attribute of R.)
(ii) Test FD (To check if $\alpha \to \beta$ holds, check if $\beta \subseteq \alpha^+$ Compute α^+ & and check if it contains β)
(iii) Compute F^+ ($\forall \gamma \in R$, find γ^+ and $\forall S \subseteq \gamma^+$ of in FD: $\gamma \to S$.)

— Canonical Cover

- Minimal set of FD's equivalent to F with no redundant dependencies nor redundant parts.

- To check if any attribute in $\alpha \to \beta$ in F is extraneous

(i) If $A \in \alpha$ is extraneous in α.
Compute $(\{\alpha\} - A)^+$ using FD's and check if it contains β. Then A is extraneous in α

(ii) If $A \in \beta$ is extraneous in β
Compute $(\alpha)^+$ using only FD's in $F' = (F - \{\alpha \to \beta\}) \cup \{\alpha \to (\beta - A)\}$

Check if α^+ contains A, then A is extraneous in β.

Ex: $F = \{A \to BC, AB \to C\}$
B is extraneous since $AB - B = (A)^+$ does contain C (B)
$F = \{A \to C, AB \to CD\}$
C is extraneous in $AB \to CD$, since in $F' = (A \to C, AB \to D)$
$(AB)^+ = \{ABDC\}$, since C is there, it is extraneous.

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

$= \{A \to BC, B \to C, AB \to C\}$ (Combine $A \to B$)
$= \{A \to BC, B \to C\}$ (A is extraneous in $AB \to C$)
$= \{A \to B, B \to C\}$ (C is extraneous, can be obtained by transitivity)

(Canonical Covers)

Ex: (Lossless - Join Decomposition)
Decomposition of R into R_1 and R_2 is lossless if atleast
$R_1 \cap R_2 \to R_1$
$R_1 \cap R_2 \to R_2$.
$R(A, B, C)$
$F = \{A \to B, B \to C\}$

__/__/____

(i) $R_1 = (A, B)$, $R_2 = (B, C)$ (no dependency list)
$R_1 \cap R_2 = \{B\}$ and $B \Rightarrow BC$
Since B is superkey for R2, lossless join ✓
(Dependency preserved) ✓

(ii) $R_1(A, B)$, $R_2(A, C)$
$R_1 \cap R_2 = \{A\}$ and $A \rightarrow AB$
Since A is superkey of R1, lossless join ✓
But since $B \rightarrow C$ is not directly available, dependency is not preserved directly. ✗

Ex:- (BCNF Decomposition)
class (course_id, title, dept_name, credits, sec_id, semester, year, building, room_number, capacity, time_slot_id)
FD: course_id → title, dept_name, credits
building, room_number → capacity
course_id, sec_id, semester, year → building, room_number, time_slot_id
Candidate key {course_id, sec_id, semester, year}

(i) course_id → title, dept_name, credits
But course_id is not a superkey in R.
Decompose class into:
course (course_id, title, dept_name, credits) → in BCNF
class1 (course_id, sec_id, semester, year, building, room_number, capacity, time_slot_id)

(ii) building, room_number → capacity
→ not superkey of class1.
Decompose into:
classroom (building, room_number, capacity)
section (course_id, sec_id, semester, year, building, room_number, time_slot_id) → in BCNF

__/__/____

– Multivalued Dependency (MVD)

- One attribute A determining multiple independent values of another attribute B ($A \twoheadrightarrow B$)

Ex: An instructor can have multiple students.
If $\alpha \rightarrow \beta$ then $\alpha \twoheadrightarrow \beta$.

– Fourth Normal Form (4NF)

- $\alpha \twoheadrightarrow \beta$ is trivial ($\beta \subseteq \alpha$, $\alpha \cup \beta = R$) (or)
- α is superkey for schema R.

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

$F = \{ A \twoheadrightarrow B$
$B \twoheadrightarrow HI$
$CG \twoheadrightarrow HJ \}$

R is not in 4NF since $A \twoheadrightarrow B$ and A is not superkey of R.

$R_1 = (A, B)$ ✓
$R_2 = (A, C, G, H, I)$ ✗
$R_3 = (C, G, H)$ ✓
$R_4 = (A, C, G, I)$ ✗

Since $A \twoheadrightarrow B$, $B \twoheadrightarrow HI$, $A \twoheadrightarrow HI$
$A \twoheadrightarrow I$

$R_5 = (A, I)$ ✓
$R_6 = (A, C, G)$ ✓ (no non-trivial MVD left)