DBMS

* collection of Interrelated data and a set of programs to access those data.

Goal: store and retrieve database info should be convenient and efficient

Applications

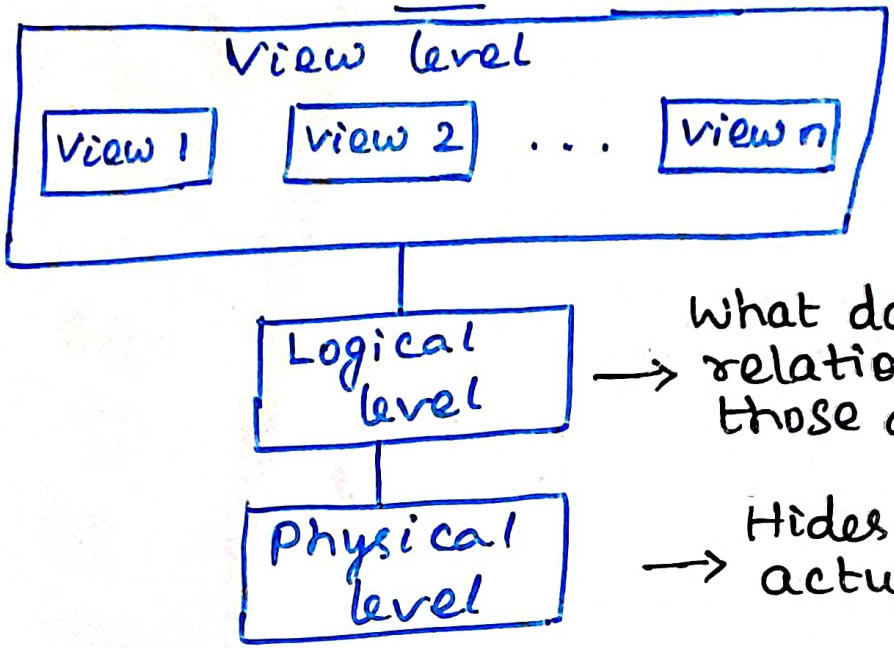
1. Enterprise Info
   - Sales (customer, product, purchase)
   - Accounting (payments, balance, receipts, assets)
   - Human Resources (employee, salary)
   - manufacturing (supply chain, tracking production of items)
   - Online retailers
2. Banking & finance
3. Universities
4. Airlines
5. Telecommunication

Purpose of Database Systems

1. Data redundancy (various copies of same data) and inconsistency
2. Difficulty in accessing data
3. Data isolation
4. Integrity problems – Data values satisfy some constraints
5. Atomicity problems – It must happen entirely (or) not at all
6. Concurrent Access Anomalies
7. Security problems

View of Data! The system hides certain details of how the data are stored and maintained.

Data Abstraction



CustID	Cname	Loanno	Amount
101	Ravi	2008	25000.00

View level (or) External level

```
Create table customerloan
(CustID    number(4)
 Cname     varchar2(20)
 Loanno    number(4)
 Amount    number(8,2));
```

Logical level (or) conceptual level

CustID	TYPE = BYTE(4), OFFSET=0
Cname	TYPE = BYTE(20), OFFSET=4
Loanno	TYPE = BYTE(4), OFFSET=24
Amount	TYPE = BYTE(8), OFFSET=28

Physical level (or) Internal level

Database Languages

Data Definition language (DDL)
↓
To specify the database schema

Data Manipulation Language (DML)

Instances and Schemas :

* The collection of information stored in the database at a particular moment is called an instance of the database.

* Overall design of the database - schema

Data model : A collection of conceptual tools for describing data, data relationships, data semantics and consistency constraints.

* classified into 4 categories

1. Relational model (Table)
2. Entity Relationship (ER) model - Objects are entities and relationship among those objects
3. Object based Data model - Java, c++, c#
4. Semi structured Data model - Data item of the same type have different sets of attributes. All data models except this, data item of the same type have same set of attributes. XML is used to represent semistructured data.

DML - Users to access data by giving Queries (or) updates

Types of Access - Retrieve, insert, delete, modify

DML

Procedural DML

↓

Users specify what data are needed and how to get those data

Non Procedural DML (or) Declarative

↓

Users specify what data are needed without specifying how to get those data.

Query: It is a statement requesting the retrieval of information

DDL: It is also used to specify consistency constraints.

1. Domain constraints 2. Referential Integrity
3. Assertion 4. Authorization

Domain constraints: A domain of possible values must be associated with every attribute

Referential Integrity: To ensure, a value that appears in one table for a given set of attributes also appears in a certain set of attributes in another relation

Assertions: A condition that the database must always satisfy. DC & RC are special forms of assertions

Authorization: To differentiate among the users for the type of access.

Relational Databases – Based on relational model.

* collection of tables (represent both data and relationships)

Table

SID	name	address
101	xyz	Salem
102	Ramu	Erode
103	Sita	Chennai

→ attributes, fields, columns

→ rows, records, tuples, data item

DDL : Create table student (SID number(5), name varchar2(20), address char(20));

DML : Select * from Student where SID = 102;

Database Design

* involves the design of the database schema.

Design process
1. To specify the data requirements of the database users
2. How the database will be structured to fulfill requirements

⇓

Designer chooses a data model (ex – relational model)

Relational model

What attributes to capture in the database

How to group these attributes to form tables

↓

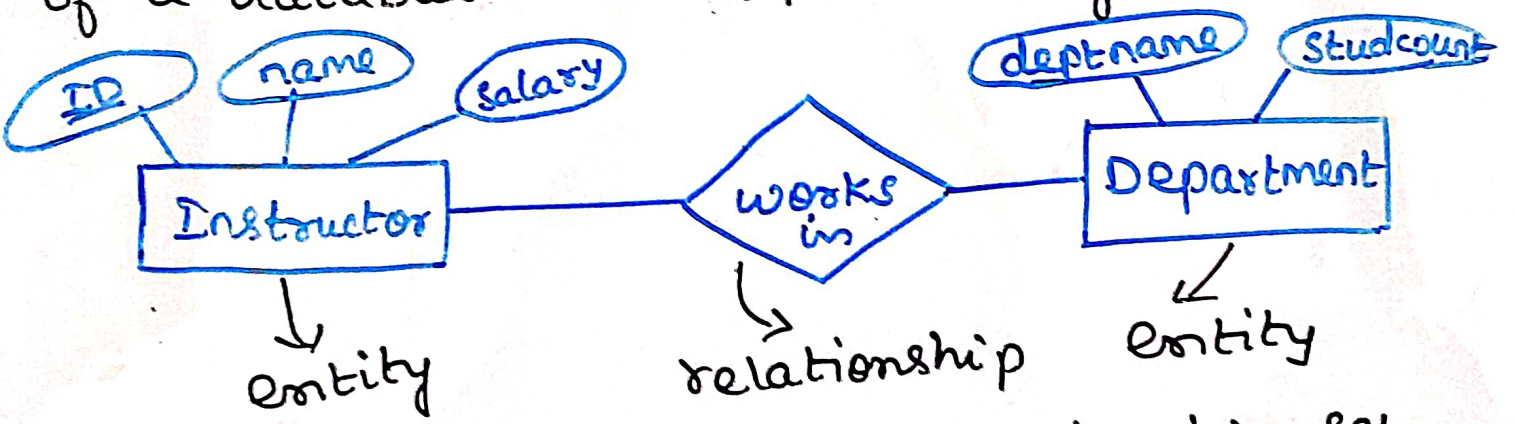
Two ways

1. Entity Relationship model (ER model)
2. Normalization (Employ a set of algorithms that takes as input the set of all attributes and generate a set of tables)

ER model (Top down approach)

* collection of entities and relationships
* Entity is a 'thing' or 'object' in the real world.
* Entity is described by a set of attributes.
* A relationship is an association among entities
* A set of all entities of the same type - entity set
* A set of all relationships of the same type - relationship set

ER Diagram - The Overall structure (schema) of a database is expressed graphically.



* Two entity sets, 1 relationship set

Normalization - (Bottom up approach)

* Eliminate redundancy
* updation anamalies

Data storage and Querying

Database system

partitioned into two modules

- Storage manager
- Query Processor

Storage manager

provide interface b/w low level data stored in the database and the application programs and Queries

* Data are stored in the disk using the file system provided by OS
* Storage manager translates DML statements into file system commands

Responsibilities	components	Data structures
Storing	Authorization and integrity manager	Data files
retrieving	Transaction manager	Data dictionary
Updating	File manager	Indices
	Buffer manager	

Authorization and Integrity manager - checks the authority of users and integrity constraints

Transaction Manager - Ensures consistency, Concurrent transactions without any conflicts

filemanager - Allocates space on disk storage and select the data structures to store the information

Buffer Manager - fetching data from disk into main memory and deciding what data to cache in main memory.

Datafiles : store the database
Data dictionary : store metadata about the structure of the database
Indices : Provide fast access to data items

Query Processor

components : DDL interpreter, DML compiler, Query evaluation engine

DDL interpreter : Interprets DDL statements and place the definition in data dictionary

DML compiler : Translates DML statements into low level instructions that the query evaluation engine understands.

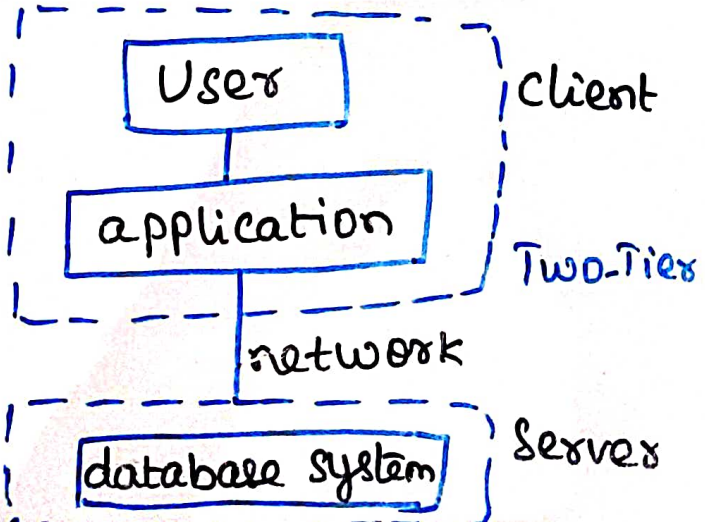
QE Engine : Executes low level instructions generated by the compiler.

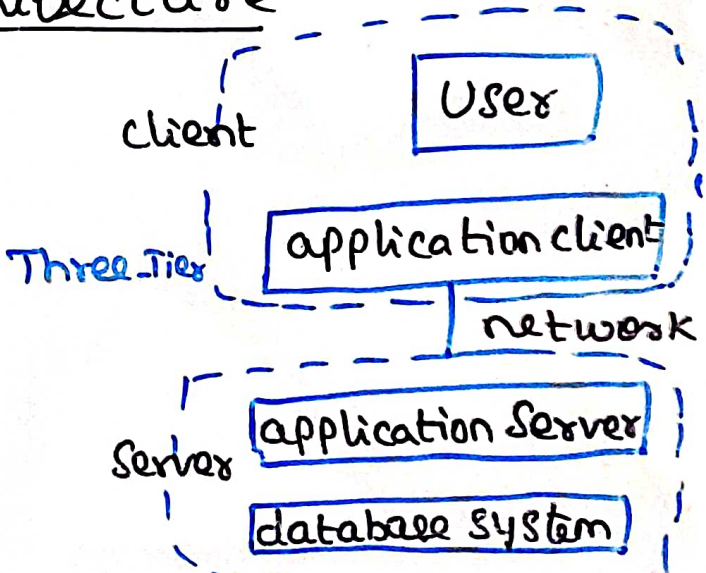
Transaction Management

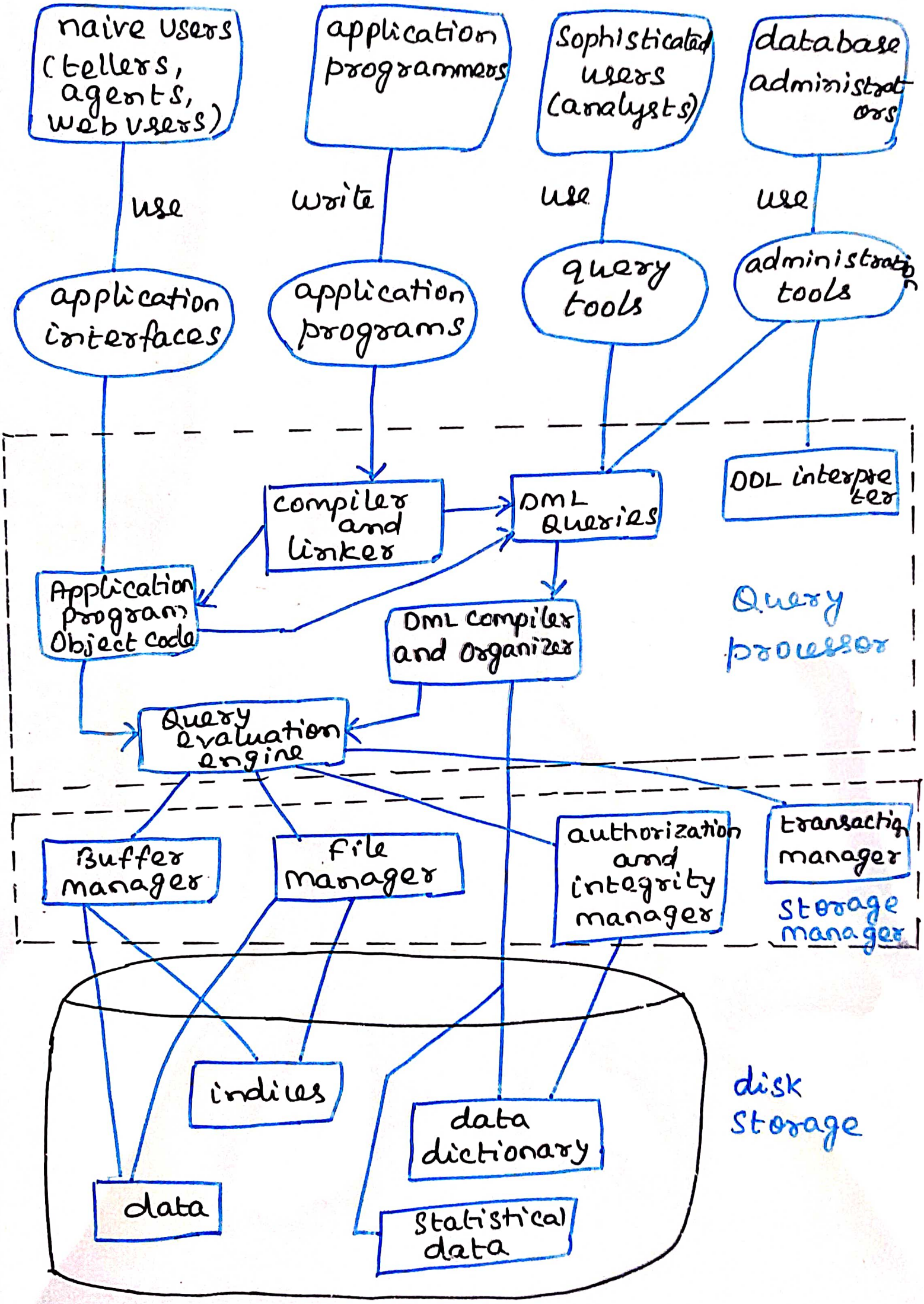
* Transaction is a collection of operations that performs in a single logical unit.

* ACID properties

Database Architecture






naive users (tellers, agents, web users)
application programmers
Sophisticated users (analysts)
database administrators
use
write
use
use
application interfaces
application programs
query tools
administration tools
compiler and linker
DML Queries
DDL interpreter
Application program Object code
DML compiler and Organizer
Query processor
Query evaluation engine
Buffer manager
file manager
authorization and integrity manager
transaction manager
Storage manager
indices
data dictionary
data
Statistical data
disk storage

Database Users and Administrators

* People who work with the database

① Naive Users, Application programmers,

③ Sophisticated Users – Without writing programs, give requests either using a database query language or using tools such as data analysis software.

④ Specialized Users – Write specialized database applications (ex) CAD, KB and Expert systems

Database Administrator

A person who has central control of both data and the programs that access those data.

functions of DBA

1. Schema definition
2. Storage structure and access method
3. Schema and physical organization modification
4. Granting of authorization for data access
5. Routine maintenance
   - Periodically backing up the database
   - Ensuring enough free disk space is available
   - Monitoring jobs running on the database
   - Ensuring the performance that is not degraded.

Relational Model

relational database – collection of tables

Instructor Relation

ID	name	dept name	Salary
1001	Ram	IT	65000
1002	Sita	CSE	75000
1003	Banu	ECE	45000

Course relation

CID	title	deptname	credits
CS401	DBMS	CSE	3
CS402	OS	CSE	3
IT401	DAA	IT	4

Student Relation

SID	name	deptname	address
IR001	XYZ	IT	Salem
IR002	abc	IT	Erode
IR003	PQR	IT	Sathy

Student (SID, name, deptname, address)

course (CID, title, deptname, credits)

Instructor (ID, name, deptname, salary)

Keys – To identify the tuples in the relation

Super Key: One (or) more attributes taken collectively to identify uniquely a tuple in the relation.

Student (SID, Sname, address, mailID, mobileno)

List of Superkeys:

{SID} {SID, Sname} {SID, Sname, address}
{mailID} {mobileno} {Sname, address}
{SID, address} {SID, mailID} {mailID, mobileno}
{address, mailID} .....
{SID, Sname, address, mailID, mobileno}.

A superkey may contain extraneous attributes.

Candidate key: If K is a super key, then no proper subset is a super key. Such minimal superkeys are called candidate keys.

{SID} {sname, address} {mailID} {mobile no}

Primary key: It is a candidate key, chosen by the database designer to identify tuples within a relation. It should be chosen such that its attribute values are never, or very rarely changed.

{SID} primary key is always underlined.

Foreign key:

Instructor (r1)

ID	name	deptname	salary
1001	Ram	IT	65000
1002	sita	CSE	75000
1003	banu	ECE	45000

(r2) department

deptname	Location
IT	IT Park
CSE	IT Park
ECE	ECE block

A relation r1, may include among its attributes, the primary key of another relation r2, then this attribute is called foreign key from r1, referencing r2

r1 is called referencing relation

r2 is called referenced relation

Number of possible superkeys in DBMS

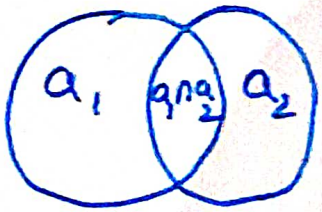
primary key ⊆ candidate key ⊆ superkey

i) Let a relation R have attributes $(a_1, a_2 \ldots a_n)$
Then the super keys $= 2^n - 1$
(ex): Student (Sid, mailid) then $2^2 - 1 = 3$

ii) Let a relation R have attributes (a_1, a_2, a_3) and a_1 is a candidate key.
Here, any superset of a_1 is the superkey.
∴ Superkeys are $= \{a_1, a_1a_2, a_1a_3, a_1a_2a_3\} = 4$
In general, a relation have 'n' attributes with one candidate key, then the number of possible superkeys are $2^{(n-1)}$

iii) Let a relation R have attributes $(a_1, a_2 \ldots a_n)$ and the candidate key is $a_1a_2a_3$ then the possible superkeys are $2^{(n-3)}$

iv) Let a relation R have attributes $(a_1, a_2 \ldots a_n)$ and the candidate keys are 'a_1', 'a_2' then the possible no/.. of superkeys are



$$|a_1 \cup a_2| = |a_1| + |a_2| - |a_1 \cap a_2|$$
$$= 2^{(n-1)} + 2^{(n-1)} - 2^{(n-2)}$$

v) Let a relation R have attributes $(a_1, a_2 \ldots a_n)$ and the candidate keys are 'a_1', 'a_2a_3' then the possible no/.. of superkeys are

$$= \text{Superkeys}(a_1) + \text{Superkeys}(a_2a_3) - \text{Superkeys}_{(a_1a_2a_3)}$$
$$= 2^{(n-1)} + 2^{(n-2)} - 2^{(n-3)}$$

Schema Diagrams: A database schema, along with primary key and foreign key dependencies, can be depicted by schema diagrams. Figure shows the schema diagram for our university organization. Each relation appears as a box, with the relation name at the top, and the attributes listed inside the box. Primary key attributes are shown underlined. Foreign key dependencies appear as arrows from the foreign key attributes of the referencing relation to the primary key of the referenced relation. Referential integrity constraints other than foreign key constraints are not shown explicitly in schema diagrams.

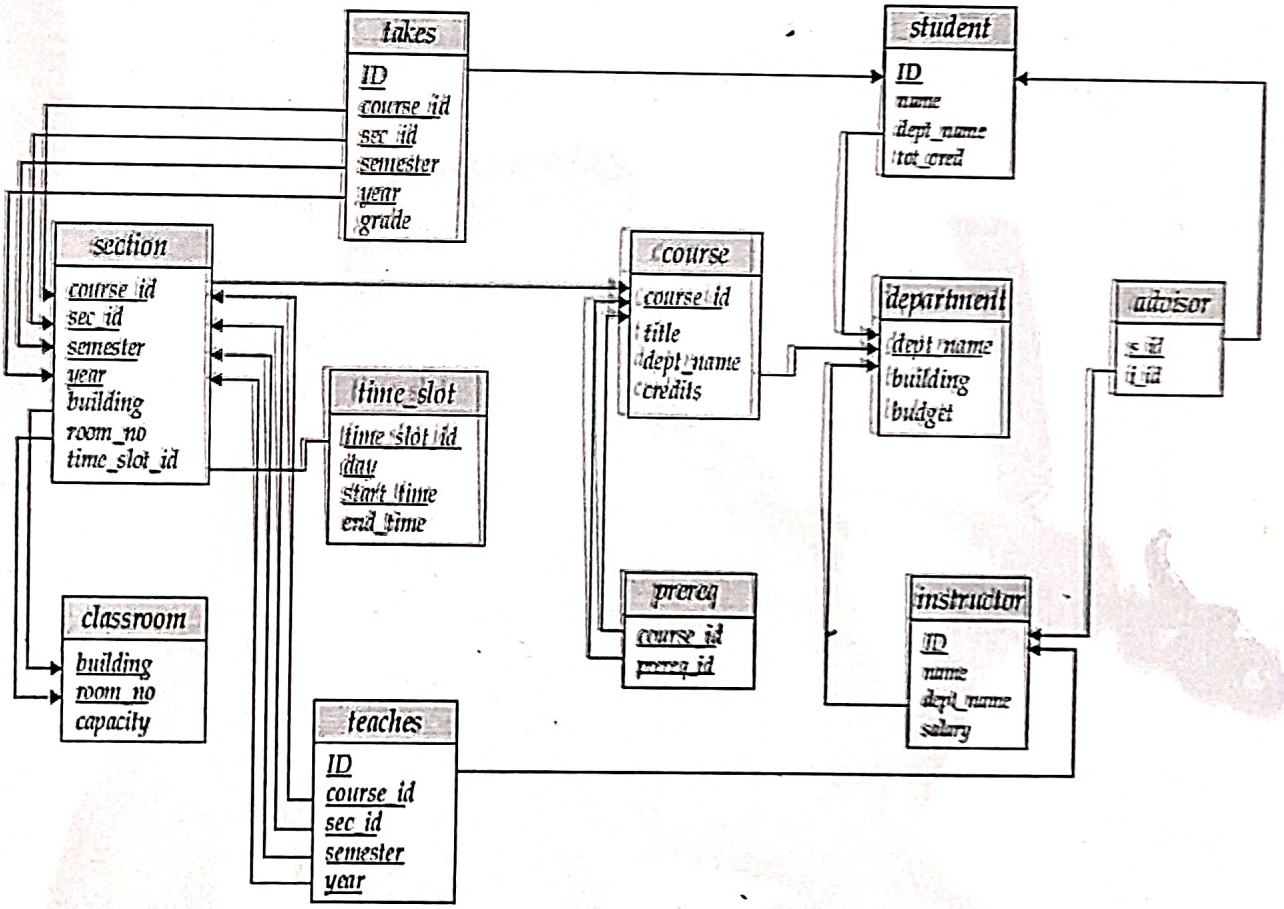


Figure 2.8 Schema diagram for the university database.

Relational operations

* A set of operations that can be applied to either a single relation (or) a pair of relations, and return a relation as an output.

* SQL is based on relational algebra.

Symbol (Name)	Example of Use
σ (selection)	find the names of employees whose salary is greater than or equal to 8000 $\sigma_{salary >= 8000}$ (employee) Selection operation return rows of the input relation that satisfy the predicate
Π (projection)	List the employee ID and salary. $\Pi_{id, salary}$ (employee) Display the attributes from all rows of the input relation.
X (cartesian product)	employee x department Display all pairs of rows from the two input relations regardless of whether (or) not they have the same values on common attributes

Relational operators

Six basic operators : Select (σ), project (Π), union ($\cup$), set difference ($-$), cartesian product ($\times$), rename (ρ)

The operators take one (or) two relations as inputs and produce a new relation as a result.

Select operation

Notation : $\sigma_p(r)$ where r is the relation
P is the selection predicate

P is connected by $\wedge$ (and), $\vee$ (or), $\neg$ (not).
Also P is connected by any one of these operators
$\neq, =, >, \geq, <, \leq$

example:

$$\sigma_{\text{branchname} = \text{'Erode'}}(\text{account})$$

Project operation

Notation : $\Pi_{A_1, A_2, \ldots A_k}(r)$ where A_1, A_2 are attribute names

The result is, it listed the k columns and erase the columns that are not listed

example:

$$\Pi_{\text{acc\#, balance}}(\text{account})$$

It list acc#, balance attributes and eliminate brancename attribute.

Union operation

Notation: $r \cup s$

for $r \cup s$ to be valid,

1. r, s must have the same arity (same number of attributes)
2. The attributes datatype must be compatible (

2nd column of r deals with the same data type of values in the 2nd column of S). 12b

example: List all customers with either an account (or) loan.

Π_{cname} (depositor) $\cup$ Π_{cname} (borrower)

Set difference

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

S

r − s :

A	B
α	1
β	1

1. r and s have same arity
2. r and s must be compatible

notation: r − s

example: List the customers who have deposits but no loans.

Π_{cname} (depositor) − Π_{cname} (borrower)

cartesian product

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

S

r × s :

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

notation: r × S

Composition of operations

Build expressions using multiple operations

(ex): $\sigma_{A=C}(r \times s)$

A	B	C	D	E
α	1	α	10	a
β	2	β	10	a
β	2	β	20	b

(ex2): $\Pi_{A,B,D}(\sigma_{A=C}(\alpha \times \beta))$

A	B	D
α	1	10
β	2	10
β	2	20

Rename operation

* Allows us to name, and therefore to refer to, the results of relational algebra expressions.

example:

$\rho_x(E)$ where it returns the expression E under the name x.

(ex): Suppose the expression E has arity n, then,

$\rho_{x(A_1, A_2, A_3 \cdots A_n)}(E)$ returns the result of expression E under the name x, and with the attributes renamed to $A_1, A_2 \cdots A_n$.

Banking database

branch (bname, bcity, assets)
customer (cname, cstreet, ccity)
account (acc#, bname, balance)
loan (loan#, bname, amount)
depositor (cname, acc#)
borrower (cname, loan#)

1) Find all loans of over Rs. 20,000.

$\sigma_{amount > 20,000}(loan)$

2) Find the loan number for each loan of an amount greater than Rs. 20000.

$\Pi_{loan\#}(\sigma_{amount > 20,000}(loan))$

3. Find the names of all customers who have a loan, an account, or both from the bank.

$\Pi_{cname}(borrower) \cup \Pi_{cname}(depositor)$

4. Find the names of all customers who have a loan at Erode branch.

$\Pi_{cname}(\sigma_{bname = "Erode"}(\sigma_{borrower.loan\# = loan.loan\#}(borrower \times loan)))$

IV III II

Step I :

loan#	bname	amount	cname	loan#

I

loan borrower

5. Find the names of all customers who have a loan at Erode branch but do not have an account at any branch of the bank.

$\Pi_{cname}((\sigma_{bname = "Erode"}(\sigma_{borrower.loan\# = loan.loan\#}(borrower \times loan))) - \Pi_{cname}(depositor))$

4. Find the names of all customers who have a loan at Erode branch.

Query 1 :

$\Pi_{cname}(\sigma_{bname = "Erode"}(\sigma_{borrower.loan\# = loan.loan\#}(borrower \times loan)))$

Query 2 :

$\Pi_{cname}(\sigma_{borrower.loan\# = loan.loan\#}((\sigma_{bname = "Erode"}(loan)) \times borrower))$

IV III I II

Additional Relational Algebra operations

1. The set intersection operation ($\cap$)

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

Query: List cname who have both deposit and loan

SQL: Select cname from depositor intersect
Select cname from borrower;

RA: $\Pi_{cname}(\text{depositor}) \cap \Pi_{cname}(\text{borrower})$

2. Natural-Join ($\bowtie$)

Query: Find the names of all instructors with the courseid of all courses they taught.

SQL: Select name, courseid from instructor
I natural join teaches;

II Select name, courseid from instructor, teaches
where instructor.ID = teaches.ID;

RA $\Pi_{name, courseid}(\text{instructor} \bowtie \text{teaches})$

The theta join is a variant of the natural join that combine selection and cartesian product into single operation. It is defined as

$$r \bowtie_\theta s = \sigma_\theta(r \times s)$$

3. Outer Join Left outer join ⟕ right outerjoin ⟖
Full outer join ⟗

4. Assignment ($\leftarrow$): Assigning parts of it to temporary relation variables.

(ex) $r \bowtie s$ can be write as, $temp1 \leftarrow r \times s$

$temp2 \leftarrow \sigma_{r.A_K = s.A_K}(temp1)$

$result = \Pi_{r \cup s}(temp2)$

$\bowtie$ natural join	employee $\bowtie$ department Display pairs of rows from the two input relations that have the same value on all attributes that have the same name
$\cup$ (union)	Π_{name} (employee) $\cup$ Π_{name} (Student) performs operation on two similarly structured tables. Display the union of tuples from the two input relations.

Introduction to SQL

SQL has several parts.

DDL, DML, Integrity constraints, view definition, Transaction control, Authorization, Embedded and Dynamic SQL

DDL: * Schema for each relation
* Types of values associated with each attribute
* Integrity constraints
* Set of indices to be maintained for each relation
* Security and authorization information

SQL Basic data types:

char(n), varchar(n), int, smallint, real, double (machine dependent), numeric(p,d), float(n)

Schema definition

create table <tablename> (A1 D1, A2 D2, ... An Dn,
<integrity constraint 1>, ...
<integrity constraint K>);

(ex)

create table department
(deptname varchar(20),
building varchar(15), budget numeric(12,2),
primary key (deptname));

Types of Integrity constraints:

Primary key: Attributes are required to be notnull and unique (no two tuples can be equal)

foreign key, not null, Unique

create table department (deptname varchar(20),
building varchar(15) Unique, budget numeric(12,2) not null,
primary key (deptname));

create table instructor (ID varchar(5) primary key,
name varchar(20) not null, deptname varchar(20),
Salary numeric(8,2), foreign key (deptname)
references department);

Delete and Drop

drop table <tablename>;
delete from <tablename>;

drop	delete
1. Entire schema is deleted from the database.	1. Schema is not deleted, but all rows are deleted.
2. Again schema is created to insert the records	2. Giving rollback command the rows can be recovered.

Truncate

Alter

alter table <tablename> add/modify A D;

(ex)

alter table emp add spousedetail varchar(20);

alter table <tablename> drop A;

But many database system do not support dropping of attributes.

Queries On a single relation

from clause — Table

Select clause — Attribute

find the names of all employees (Query)

select name from employee;

To avoid duplicates use distinct keyword after select.

Select distinct deptname from employee;

To allow duplicates use all keyword after select

Select all deptname from employee;

Select clause with arithmetic Operators (+, −, *, /)

Select eid, ename, salary * 1.1 from employee;

Select clause with logical Operators (always with where)

<, <=, >, >=, =, <>

Select ename from employee where deptname = 'computer' and salary >= 10000;

Queries on Multiple relations

Select clause – attributes
from clause – tables
where clause – predicate

Sequence of steps followed in SQL Queries on multiple relations:

1. Generate a Cartesian product of the relations listed in the from clause.
2. Apply the predicates specified in the where clause.
3. For each tuple in the result of step 2, display the attributes specified in the select clause.

Department (deptname, building, budget)
course (courseid, title, deptname, credits)
instructor (ID, name, deptname, salary)
teaches (ID, courseid, secid, semester, year)

Instructor

ID	name	deptname	salary
IT02	Nalini	IT	90,000
IT09	devi	IT	40,000
ECE32	Ajay	ECE	25,000
SH12	Senthil	SH	35,000
CSE02	Raji	CSE	95,000

Course

courseID	title	deptname	credits
CS401	DBMS	CSE	3
IT402	DAA	IT	3
EC404	ADC	ECE	3
IT403	JDP	IT	3
CS405	OS	CSE	3

teaches

ID	courseid	secid	semester	year
CSE02	CS401	A	4	2
IT09	IT402	A	4	2
ECE32	ECE404	B	4	2
IT02	CS405	B	4	2

Query: find the faculty names and courseids of the courses they taught.

Step1 : cartesian product of Instructor & Teaches

Ins.ID	name	deptname	salary	teaches ID	courseid	secid	semes	ye
IT02	Nalini	IT	90000	CSE02	CS401	A	4	2
IT02	Nalini	IT	90000	IT09	IT402	A	4	2
IT02	Nalini	IT	90000	ECE32	ECE404	B	4	2
IT02	Nalini	IT	90000	IT02	CS405	B	4	2
IT09	devi	IT	40000	CSE02	CS401	A	4	2
IT09	devi	IT	40000	IT09	IT402	A	4	2
IT09	devi	IT	40000	ECE32	ECE404	B	4	2
IT09	devi	IT	40000	IT02	CS405	B	4	2
ECE32	Ajay	ECE	25000	CSE02	CS401	A	4	2
ECE32	Ajay	ECE	25000	IT09	IT402	A	4	2
ECE32	Ajay	ECE	25000	ECE32	ECE404	B	4	2
ECE32	Ajay	ECE	25000	IT02	CS405	B	4	2

Ins.ID	name	deptname	Salary	teaches. ID	course id	secid	sem	year
SH12	Senthil	SH	35000	CSE02	CS401	A	4	2
SH12	Senthil	SH	35000	IT09	IT402	A	4	2
SH12	Senthil	SH	35000	ECE32	ECE404	B	4	2
SH12	Senthil	SH	35000	IT02	CS405	B	4	2
CSE02	Raji	CSE	95000	CSE02	CS401	A	4	2
CSE02	Raji	CSE	95000	IT09	IT402	A	4	2
CSE02	Raji	CSE	95000	ECE32	ECE404	B	4	2
CSE02	Raji	CSE	95000	IT02	CS405	B	4	2

Step 2: Apply the predicate

Select name, courseid from instructor, teaches where instructor.Id = teaches.ID (predicate)

Ins.ID	name	deptname	Salary	teaches. ID	course id	secid	semester	year
IT02	Nalini	IT	90000	IT02	CS405	B	4	2
IT09	devi	IT	40000	IT09	IT402	A	4	2
ECE32	Ajay	ECE	25000	ECE32	ECE404	B	4	2
CSE02	Raji	CSE	95000	CSE02	CS401	A	4	2

Step3: Display the attributes specified in the select clause

name	courseid
Nalini	IT02
devi	IT09
Ajay	ECE32
Raji	CSE02

Natural join:

cartesian product – It take two relations, which concatenates each tuple of the first relation with every tuple of the second.

Natural join – It consider only those pairs of tuples with the same value on those attributes that appear in both relations

ID	name	deptname	salary	courseid	secid	semes	year
IT02	Nalini	IT	90000	CS405	B	4	2
IT09	devi	IT	40000	IT402	A	4	2
ECE32	Ajay	ECE	25000	ECE404	B	4	2
CSE02	Raji	CSE	95000	CS401	A	4	2

cartesian product = 5 × 4 = 20 tuples

natural join = 4 tuples

Display order of natural join:

* Attributes common to both relations (appear once)
* Attributes unique to first relation
* Attributes unique to second relation

Select name, courseid from instructor natural join teaches ;

Syntax for natural join:

Select $A_1, A_2, \ldots A_n$ from r_1 natural join r_2 natural join r_3 ... natural join r_m where P ;

List the names of instructors along with the titles of courses that they teach. I-2

```
Select name, title from instructor
natural join teaches, course where
teaches.courseid = course.courseid;
```

Steps:

1. Natural join b/w instructor & teaches (4 tuples)
2. Cartesian product b/w Step 1 & course (20 tuples)
3. Apply predicate condition (Only 4 tuples are selected)
4. Display the selected attributes.

The following SQL query does not give the same result

```
Select name, title from instructor
natural join teaches natural join course;
```

ID	name	deptname	salary	courseid	...
IT02	Nalini	IT	90000	CS405	
IT09	devi	IT	40000	IT402	
ECE32	Ajay	ECE	25000	ECE404	
CSE02	Raji	CSE	95000	CS401	

course ID	title	dept name	c
CS405	OS	CSE	
IT402	DAA	IT	
ECE404	ADC	ECE	
CS401	DBMS	CSE	

deptname is a common attribute, so it give only 3 tuples.

This Query Omit the tuples where the instructor teaches a course in the dept, Other than the instructors own dept course.

name	title
devi	DAA
Ajay	ADC
Raji	DBMS

* To provide the benefit of natural join while avoiding the danger of equating attributes erroneously, SQL provides a form of natural join construct with join ... using construct.

Select name, title from (instructor natural join teaches) join course using (courseid); ← column should be equated

Additional Basic Operations

Rename : renaming the attributes of a result relation.

Syntax : oldname as newname

ex Select name as instructorname from emp;

Select T.name , S.courseid from instructor as T , teaches as S where T.ID = S.ID;

String Operations

1. Length 2. Upper 3. lower 4. trim

pattern matching

percent (%) : The % character matches any substring

Underscore (_) : The _ character matches any character

(ex) 'Intro%' — Any string begin with Intro

'%Comp%' — Intro to computer science

"___" matches any string of exactly 3 characters

"___%" matches any string of atleast 3 characters

Like, not like — comparison of strings.

```
select deptname from department where
    building like '%y.park%';
```

escape — The escape character \ is used immediately before a special pattern character to indicate that the special pattern character is to be treated like a normal character.

(ex) like 'ab\@cd'

Ordering the display of Tuples

Order by clause makes the records of a query in sorted order.

```
select name from instructor where
    deptname = 'IT' Order by name;

select name from instructor order by
    Salary desc , name asc ;
```

Attribute specification in Select clause

```
select * from emp;
select instructor.* from instructor, teaches
where instructor.ID = teaches.ID
```

Between .. and

(or) Select name from instructor where salary between 90000 and 100000;

Select name from instructor where salary not between 90000 and 100000;

→ This Query is equal to,

Select name from instructor where Salary <= 100000 and Salary >= 90000;

Problem: find the instructor names and course ids they taught, for all instructors of IT department.

Use cartesian product,

Select name, courseid from instructor, teaches where instructor.ID = teaches.ID and deptname = 'IT';

* SQL Permits to use $(v_1, v_2 \ldots v_n)$ to denote a tuple of arity n containing values $v_1, v_2 \ldots v_n$.

* The comparison operators can be used on tuples and the ordering is lexicographically.

(ex) $(a_1, a_2) <= (b_1, b_2)$ is true if $a_1 <= b_1$ and $a_2 <= b_2$

So, apply this concept, the previous Query can be written as,

Select name, courseid from instructor, teaches where (instructor.ID, deptname) = (teaches.ID, 'IT');

Set Operations

SQL Operations are union, intersect, except
Relational algebra operations are ∪, ∩, −

depositor

cname	deposit no	acc no
John	1001	AC01
Sita	1002	AC02
vijay	1003	AC10
Ram	1004	PC14

loan

cname	loan no	acc no
John	2001	AC01
Reema	2002	AC03
Manju	2003	AC11
vijay	2004	AC10

Union: It merges the o/p of two (or) more queries into a single set of rows and column.

Query: Find all customers having a loan, deposit or both at the bank.

```
Select cname from borrower union
Select cname from depositor;
```

Union eliminates duplicates, to retain duplicates,

```
Select cname from borrower union all
Select cname from depositor;
```

Restrictions on using a Union

1. Number of columns should be same.
2. Datatype of the column must be same
3. Union can not be used in Subqueries.
4. Aggregate functions can not be applied in

Intersect : It returns common records from the output of both queries.

Query: Find all customers who have both deposit and loan.

Select cname from depositor intersect
Select cname from borrower; Intersect all

Except: It is also called minus, display rows that are in first table but not in second table.

Query: find all customers who have an deposits but no loan at the bank.

Select cname from depositor minus
select cname from borrower;

Null Values: SQL allows the use of null values to indicate absence of information about the value of an attribute. Use the special keyword null in a predicate to test for null value.

Customer

Accno	cname	cphno
AC01	John	220102
AC02	Sita	243812
AC03	Reena	281011
AC10	Vijay	
AC11	manju	271012
AC14	Ram	

Query: find the customers who have no phones.

Select cname from Customer where Cphno is null;

The predicate is not null is also used,

Select cname from customer where cphno is not null;

Aggregate functions

Aggregate functions are functions that take a collection of values as input and return a single value. SQL offers five built-in aggregate functions.

avg, min, max, sum, count

(ex) avg

```
Select avg(salary) from instructor
    where deptname = 'IT';
```

Select avg(salary) as average_salary from instructor where deptname = 'IT';

Count

```
Select count(*) from course;
```

Query: find the total number of instructors who teach a course in even semester.

```
Select count(distinct ID) from teaches
    where semester = 'even';
```

note: Distinct can not be combined with count(*)

min:

```
Select min(salary) from instructor;
```

max:

```
Select max(salary) from instructor;
```

Sum:

```
Select sum(salary) from instructor;
```

Aggregation with grouping!

Tuples with the same value on one (or) more attributes in the group by clause are placed in one group.

Query! Find the average salary of each department

Select deptname, avg (salary) from instructor
group by deptname;

Query steps! 1) instructor relation grouped by deptname.

2) aggregate function is applied for each group.

3) To ensure the attributes that appear in the select statement are aggregated attributes present in the groupby clause.

If any attribute present in the select clause but not present in group by clause is treated as erroneous.

(ex) Select deptname, ID, avg (salary) from instructor group by deptname;

Having! This clause tells SQL to include only certain groups produced by the group by clause in the query result.

Having clause is equivalent to the Where clause and is used to specify the search criteria (or) search condition when group by clause is specified.

(ex) Find the departments whose average salary of the department is more than 40,000.

```
Select deptname, avg(salary) from instructor
group by deptname having avg(salary) > 40,000;
```

Steps for having clause:

1. from clause is evaluated.
2. If where clause is present, where clause is applied on the from clause.
3. Rows satisfying the where condition are then placed into groups by the group by clause.
4. Having clause is applied to each group. The groups that do not satisfy the having clause are removed.
5. Finally select clause is applied to generate the tuples of the query.

Aggregation with Null values

Aggregate functions except count(*) ignore null values.

The count of an empty collection is defined to be 0, all other aggregate functions return a value of null when applied on an empty relation.

Nested SubQueries:

A subquery is a select-from-where expression is nested within another query.

Use of Subqueries:

1. Perform tests for set membership
2. Set comparisons
3. determine set cardinality.

Set membership:

SQL uses in and not in constructs for set membership tests.

in: The in connective tests for set membership, where the set is a collection of values produced by a select clause.

Book

Title	Authorname	Pubname	Year
Oracle	Arora	PHI	2004
DBMS	Korth	TMH	2008
Unix	Kapoor	SciTech	2000
DDBMS	Rama	Technical	2004

Query: Display the title, author, pubname of all books published in 2000, 2002, 2004

```
Select title, authorname, pubname from book
where year in ('2000', '2002', '2004');
```

Query: Give the details of author who have published book in the year 2008.

author (title, authorname, country)

Select * from author where authorname in (select authorname from book where year = '2008');

Not in: This test for the absence of set membership.

Query: Display title, author, pubname of all books except those which are published in the year 2002, 2004 and 2005.

Select title, authorname, pubname from book where year not in ('2002', '2004', '2005');

Query: Get the titles of all books written by authors not living in India.

Select title from book where authorname not in (select authorname from author where country = 'India');

Set comparison: Nested subQueries are used to compare sets. SQL allows various comparison operators such as <, <=, >, >=, =, <>.

phrases: greater than at least one is represented in SQL as > some

= some is identical to in

greater than all is represented in SQL as >all. <>all is identical to not in.

Display the title of books that have price greater than all of the books published in the 2004.

Select title from price where unitprice > all (select unitprice from price where year = '2004');

Test for Empty Relations

Exists is a test for non empty set. It is represented in the form exists (select .. from). Such an expression evaluates to true only if the result of evaluating the subquery represented by the (select ... from) is non empty.

book

bookID	Title	Author name	Pub name	Year
1001	Orale	Arora	PHI	2004
1002	DBMS	Korth	TMH	2004
1003	ADBMS	Sinha	Technical	2003
1004	C	Samy	Oxford	2000
1005	Java	Kapoor	Wiley	2004

order

Order ID	bookID	order date	qty	Price
1	1001	1.12.12	50	400
2	1002	3.12.12	100	500

Query: Get the names of all books for which order is placed.

Select title from book where exists (select * from order where book.bookID = order.bookID);

{1001, 1002, 1003, 1004, 1005} exists {1001, 1002}

Check (same as natural join)

Select title from book natural join order;

Not exists! Similar to exists.

Query: Display the titles of books for which order is not placed.

```
Select title from book where not exists
    (select * from order where book.bookID
                              = order.bookID);
```

check (cartesian product with where)

```
select title from book, order where
  book.bookID <> order.bookID;
```

Complex Queries

Queries which is difficult (or) impossible to write in a single SQL block are called as complex Queries.

There are two ways to express a complex query.

1. Derived relations 2. With clause

Derived relations: find the average account balance of those branches where the average account balance is greater than 10,00,000.

```
Select bname, avg(balance) from account
  group by bname having avg(balance) >
                                10,00,000;
```

In derived relations, a sub query is used in the from clause.

```
Select bname, avg(balance) from
   (select bname, avg(balance) from account
      group by bname) as bavg(bname, avg(balance))
where avg(balance) > 1000000;
```

Account

Accno	bname	balance
1001	KECnagar	3000000
1002	Erode	50000
1003	KECnagar	1000000
1004	Salem	5000000
1005	Erode	10000
1006	KECnagar	2000000

bavg

bname	avg(balance)
KECnagar	3000000
Erode	30000
Salem	5000000

find the maximum across all branches of the total balance at each branch.

```
Select max(totalbalance) from
   (select bname, sum(balance) from
      account group by bname) as
      branchtotal (bname, totalbalance);
```

Oracle do not support renaming of the result table in the from clause.

Steps in Oracle:

1. Create table branchtotal as select bname, sum(balance) as netbalance from account group by bname;
2. Select max(netbalance) from branchtotal;

Nested subqueries in the from clause can not use correlation variable (table alias, tuple variable). However, SQL provides to use correlation variable using **Lateral** keyword.

faculty(id, name, dept, salary)

Query: To print the names of each faculty along with their salary and average salary in their department.

```
Select name, salary, avg(salary) from
faculty F1, lateral (select avg(salary)
        from faculty f2 where
        f1.dept = f2.dept);
```

With Clause: It is used to define a temporary relation whose definition is available only to the query in which the with clause occurs.

Account

Accno	bname	balance
1001	KECNagar	3000000
1002	Erode	50000
1003	KECnagar	1000000
1004	Salem	5000000
1005	Erode	10000
1006	KECnagar	2000000
1007	Erode	5000000
1008	Salem	5000000

List the accounts who have maximum balance.

Consider the faculty table,

fid	fname	deptname	salary
1	Anbu	CSE	1,30,000
2	babu	CSE	50,000
3	chitra	CSE	60,000
4	devi	IT	1,30,000
5	Elango	IT	80,000
6	fathima	ECE	60,000
7	Geetha	ECE	40,000
8	Hari	ECE	20,000

Write SQL statements for the following queries.

1. Find the average salary of the faculty.
2. Find the average salary for each department.
3. Find the faculty who earn more than average salary of the faculty.
4. Find the faculty who earn more than average salary in their department.
5. find the numbes of faculty in each department.
6. Find the average salary of IT department.

SQL statements

1. select avg(salary) from faculty;
2. select deptname, avg(salary) from faculty group by deptname;
3. select fid, fname from faculty where salary > (select avg(salary) from faculty);

inner sub query (Always execute only one)

Outer query execute for all rows

Correlated Sub query

* A subquery that uses a correlation name in where clause from an outer query is called correlated sub query.

Syntax:

alias name

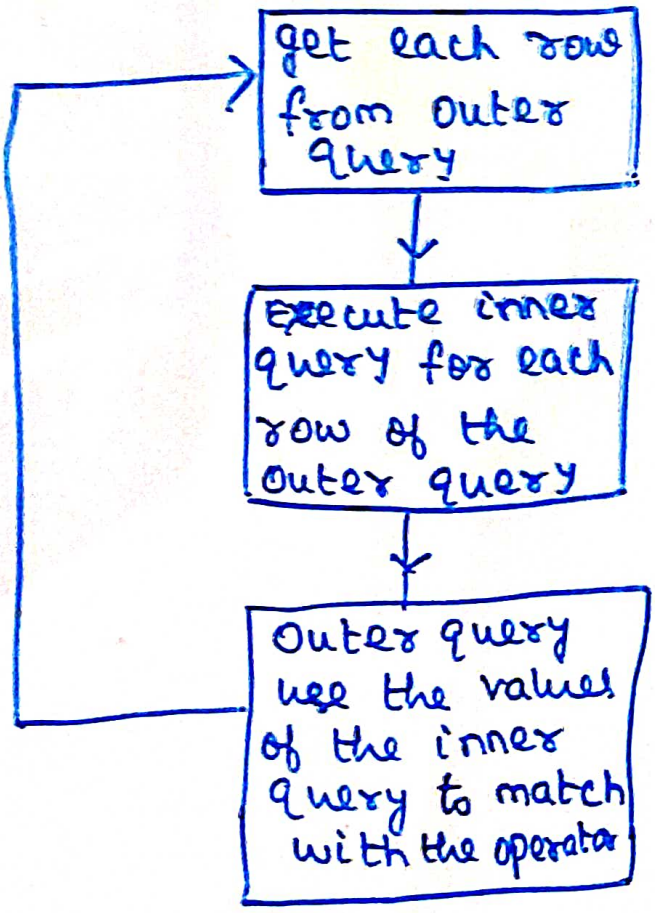
Select col1, col2 ... from table <correlation name> where col <operator> (Select col1, col2 from table where col1 = <correlation name> . col1);

SQL statement (4th question)

Select fname, deptname, salary from faculty F where salary > (select avg(salary) from faculty where deptname = F. deptname);

↑ inner sub query use correlation name of Outer query.

Flow chart for correlated subquery:



Difference blw Subquery & correlated Subquery

Subquery	correlated subquery
1. Inner subquery execute only once for the entire rows of the Outer subquery	Inner subquery execute once for each row of the Outer sub query
2. execution time is low	execution time is high.

* The key words exists, not exists also used in the correlated subquery.

Select col1, col2 ... from table <correlation name> where exists (select col1, col2 from table where col1 = corrname. col1);

5. Find the number of faculty in each department.

Select deptname, count(*) from faculty group by deptname;

6. Select avg(salary) from faculty where deptname = 'IT';

Scalar Subquery

A subquery that selects one column (or) expression as input and it return only one row containing a single attribute is called scalar subquery.

* Scalar subquery allows to treat the output of a subquery as a column within a select statement.

(ex) consider two table department and faculty.

department

did	deptname
1	CSE
2	IT
3	ECE

Apply scalar subquery to find the number of faculty in each department.

```
select deptname, (select count(*)
from faculty where faculty.deptname
= department.deptname) as noffaculty
from department;
```

scalar subquery →

Scalar subquery determine how many rows in the department table contain faculty corresponding to each row in the faculty table.

Use of scalar subqueries

1. Aggregate from multiple tables
2. Inserting into tables, based on values from other tables.

(ex) create table summary (sum_sal number(5), max_sal number(5), min_sal number(5), avg_sal number(5));

insert into summary (sum_sal, max_sal, min_sal, avg_sal) values ((select sum(salary) from faculty), (select max(salary) from faculty), (select min(salary) from faculty), (select avg(salary) from faculty));

```
with maxbalance(value) as
    select max(balance) from account
    select accno from account, maxbalance
    where account.balance = maxbalance.value;
```

Scalar Subqueries:

A subquery returns only one tuple containing a single attribute is called scalar subquery.

Query: List all departments along with the number of instructors in each department.

```
Select deptname, (select count(*) from
    instructor where department.deptname =
    instructor.deptname) as numofinstructors
    from department;
```

Modification of the Database

deletion, insertion, updation

Deletion:
```
delete from emp;
delete from emp where deptname = 'IT';
delete from emp where salary < (select avg(salary)
                                from emp);
```

insertion:
```
insert into course values ('CS401', 'DBMS',
                           'CSE', 4);
insert into course(courseid, coursename) values
                  ('CS401', 'DBMS');
insert into emp select sid, sname, 18000 from
        student where sname = 'Ravi';
```

Update:

```
update emp set salary = salary * 2;
update emp set salary = salary + 10000 where
                        salary < 70,000;
```

Give 5 percent salary rise to employee whose salary is less than average.

```
update emp set salary = salary * 1.05
 where salary < (select avg(salary) from emp);
```

Query: Raise the salary 3% for employee whose salary is less than 10000 and raise the salary 5% for others.

```
update emp set salary = salary * 1.03
   where salary <= 10000;
update emp set salary = salary * 1.05
   where salary > 10000;
```

SQL provides a case construct to perform both the updates in a single statement.

```
update emp set salary = case
      when salary <= 10000 then salary * 1.03
      else salary * 1.05
end;
```

Syntax:

```
case
when pred1 then result1
when pred2 then result2
...
when predn then resultn
else result0
end;
```

employee (empname, street, city)
works (empname, compname, salary)
company (compname, city)
manages (empname, managername)

1) Find the names of all employees who work for 'first bank corporation'.

Select empname from works where compname = 'first bank corporation';

2) Find the names and cities of residence of all employees who work for first bank corporation.

Select e.empname, city from employee e, works w where e.empname = w.empname and w.compname = 'first bank corporation';

3) Find the names, street and cities of all employees who work for first bank corporation and earn more than 10,000.

a) Select e.empname, street, city from employee e, works where employee.empname = works.empname and salary > 10,000 and compname = 'first bank corporation';

b) Select * from employee where empname in (Select empname from works where compname = 'firstbank corporation' and salary > 10,000)

4) Find all employees in the database who live in the same cities as the companies for which they work.

a) Select empname from employee natural join works natural join company;

b) Select e.empname from employee e, works w, company c where e.empname = w.empname and e.city = c.city and w.compname = c.compname;

5) Find all employees in the database who do not work for first bank corporation.

1) Select empname from works where compname <> 'first bank corporation';

2) Select empname from employee where empname not in (select empname from works where compname = 'firstbank corporation';

6) Find all employees in the database who earn more than every (greater than all) employee of small bank corporation.

Select empname from works where salary > all (select salary from works where compname = 'small bank corporation');

7) Find all employees who earn more than the average salary of all employees of their company.

Select empname from works T where salary > (select avg(salary) from works S where T.compname = S.compname);

8) Find the company that has the smallest payroll.

Select compname from works group by compname having sum(salary) <= all (select sum(salary) from works group by compname;

Intermediate SQL

Join is a query in which data is retrieved from two (or) more table. It matches data from two (or) more tables, based on the values of one (or) more columns on each table.

Need for Join!

1) To make comprehensive data analysis, assemble data from several tables.

2) To reduce data redundancy and improve data independence, relational model allow to partition the data and put into different tables.

3) To perform adhoc queries.

Different types of Joins

Inner Join, Outer Join, Natural Join

* Normal Joins are always called inner join.

Inner Join – Use join .. using clause
(or) Use join .. On clause

(ex) Select * from student join takes using(ID);

Select * from student inner join takes using(ID);

Keyword – inner is optional

natural join is equivalent to natural inner join.

Difference b/w inner join, natural join

* In natural join, common attribute appears only once.
* In inner join, common attribute appears twice.

Use join ... On clause for Inner Join!

Select * from student join takes on

Student.ID = takes.ID

On condition allows a general predicate over the relations being joined. The keyword on replace the keyword where.

Reasons for using On clause!

1. It provide richer class of join conditions than natural join
2. Query is readable by humans, if the join condition is specified in the on clause and the rest of the conditions appear in the where clause.

Outer Join! When tables are joined using inner join, rows which contain matching values in the join predicate are returned. Sometimes we require both matching and non matching rows returned for the tables that are being joined. This kind of an operation is called as Outer join.

An Outer join is an extended form of inner join. In this, the rows in one table having no matching rows in the other table will also appear in the result table with nulls.

Types of Outer join

Left Outer, Right Outer, full Outer

Left Outer Join:

The Left Outer join returns matching rows from the tables being joined, and also non-matching rows from the left table in the result table and places null values in the attributes that come from the right table.

Emp

empid	city
A1	newyork
A2	null
A3	chicago
A4	chicago
A5	paris

customer

custid	city
B1	newyork
B2	newyork
B3	null
B4	chicago
B5	moscow

emp.empid	emp.city	cust.custid	customer.city	
A1	newyork	B1	newyork	inner join output
A1	newyork	B2	newyork	
A3	chicago	B4	chicago	
A4	chicago	B4	chicaga	
A5	paris	null	null	Unmatched rows
A2	null	null	null	

```
select emp.empid, emp.city, customer.custid,
customer.city from emp, customer
where emp.city = customer.city (+);
```

Select * from emp natural left outer join customer;

city	empid	custid

Right Outer Join: It return matching rows from the tables joined, and also non-matching rows from the right table in the result and places null values in the attributes that comes from the left table.

emp

empid	city
A1	Newyork
A2	Null
A3	Chicago
A4	Chicago
A5	Paris

customer

custid	city
B1	New York
B2	New York
B3	Null
B4	Chicago
B5	Moscow

emp. empid	emp. city	customer. custid	customer. city	
A1	Newyork	B1	Newyork	inner join
A1	Newyork	B2	Newyork	
A3	Chicago	B4	Chicago	
A4	Chicago	B4	Chicago	
Null	Null	B5	Moscow	Right outer join
Null	Null	B3	Null	

Select emp.empid, emp.city, customer.custid, customer.city from emp, customer where emp.city(+) = customer.city;

(or)

Select * from emp natural right outer join customer;

city	empid	custid

(+) symbol is next to the column which needs to be expanded to include null values.

full outer join! It is a combination of both left and right outer join types. After the operation computes the result of the inner join, it extends with nulls those tuples from the left hand side table that did not match with any from right hand side table and adds them to the result table.

Similarly, it extends with nulls, those tuples from the right hand side table that did not match with any tuples from the left hand table and adds them to the result table.

Emp

empid	city
A1	newyork
A2	Null
A3	chicago
A4	chicago
A5	Paris

customer

custid	city
B1	newyork
B2	newyork
B3	Null
B4	chicago
B5	Moscow

emp. empid	emp. city	customer. custid	customer. city	
A1	newyork	B1	newyork	inner join output
A1	newyork	B2	newyork	
A3	chicago	B4	chicago	
A4	chicago	B4	chicago	
A5	Paris	null	null	Left outer join
A2	null	null	null	
Null	null	B5	Moscow	right outer join
Null	null	B3	null	

Select * from emp natural full Outer join customer;

city	empid	custid

On clause with Outer Join!

Select * from emp natural left Outer join customer on emp.city = customer.city;

(ie) the common attribute appears twice in the result table.

empid	empcity	custid	custcity

Join types and conditions

Join types
inner join
left outer join
right outer join
full outer join

Join conditions
natural
on <predicate>
using $<A_1, A_2, .. A_n>$

Normal joins are always called inner join. The keyword inner is optional.

Select * from student join courses using (ID);

↓ is same

Select * from student inner join courses using (ID);

Similary natural join is equivalent to natural inner join

Views: A view is a object that gives the user, a logical view of data from an underlying table (or) tables.

Reasons for creating views

* Simplifies Queries
* provides security
* Can be queried as a base table
* View is a virtual table, it is not precomputed and stored so it provide accurate results

Definition: Any table that is not a part of the logical model, but it is made visible to a user as a virtual relation is called a view.

Syntax: create view v as <query expression>;

(ex) create view faculty as select ID, name, deptname from instructor;

Select * from faculty; (To see the tuples in the faculty)

Materialized views

* Certain database systems allow view relations to be stored, but they make sure that, if the actual relations used in the view definition change, the view is kept upto-date. Such views are called materialized views.

* The process of keeping the materialized view up-to-date is called materialized view maintenance.

Update of a view:

Views can also be used for data manipulation (ie) the user can perform insert, update and delete operations on the view.

Updatable views – views on which data manipulation can be done

Readonly views – views do not allow for data manipulation

When the user give a view name in the update, insert or delete statement, the modifications of the data will be passed to the underlying table.

For the view to be updatable, it should meet following criteria:

1. The view must be created on a single table
2. The primary key column of the table should be included in the view.
3. Aggregate fns cannot be used in the select statement
4. The select statement used for creating a view should not include distinct, group by (or) having clause.

5. The select statement used for creating a view should not include subqueries.
6. It must not use constant, strings (or) value expressions like salary/6
7. Any attribute not listed in the select clause can be set to null; (ie) it does not have a not null constraint and is not part of a primary key.

Destroying a view

A view can be dropped by using drop view command.

(ex) drop view <viewname>;
drop view v_book;

Transactions

A transaction consists of a sequence of query and/or update statements. The SQL standard specifies, a transaction begins implicitly when an SQL statement is executed. Either commit (or) rollback statements must end the transaction.

Commit work: It commits the current transaction. (ie) it makes the updates performed by the transaction become permanent in the database. After the transaction is committed, a new transaction is automatically started.

Rollback work: It causes the current transaction to be rolled back. (ie) it undoes all the updates performed by the SQL statements in the transaction. Thus, the database is restored to what it was before the first statement of the transaction was executed.

Keyword work is optional.

Integrity constraints

It ensure that changes made to the database by authorized users do not result in a loss of data consistency. Thus integrity constraints guard against accidental damage to the database.

(ex) 1. An instructor name cannot be null.
2. No two students can have same ID.
3. Minimum salary should be 8000.

IC are created using 1. create table (or)
2. Alter table add commands.

↓

* When such a command is executed, the system first ensures that the relation satisfies the specified constraint. If it does, the constraint is added to the relation. If not, the command is rejected

Constraints on a Single Relation

1. Primary key 2. not null 3. Unique
4. check <predicate>

Not null : It prohibits the insertion of a null value for the attribute.

(ex) name varchar(20) not null
budget number(12,2) not null

Unique : No two tuples in the relation can be equal on the listed attributes.

(ex) unique ($A_1, A_2, \ldots A_n$)

Unique constraint used for creating a candidate key. Candidate key attributes are permitted to be null unless they have explicitly been declared to be not null. A null value does not equal to any other value.

Check clause : When applied to a relation declaration, the check clause specifies a predicate P must be satisfied by every tuple in a relation.

(ex) check (budget > 0) — It ensure the value of budget is non negative

* check clause permits attribute domains to be restricted.

(ex) check (color in ('red', 'green', 'blue'))

constraints on a multiple relation

Referential Integrity

* To ensure that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation. This condition is called referential integrity.

(ex) Create table department (deptname varchar(20), building varchar(15), budget number(12,2) check (budget > 0), primary key (deptname));

Create table instructor (ID varchar(5), name varchar(20) not null, deptname varchar(), salary number(10,2) check (salary > 25000), primary key (ID), foreign key (deptname) references department);

On delete cascade, On delete set null

Create table instructor (...
foreign key (deptname) references department
on delete cascade); on update cascade
on delete set null
on delete set default

On delete cascade associated with the foreign key declaration, if a delete of a tuple in referenced table (parent) 'cascades' to the referencing table (child), deleting the tuple that refers to the referenced table.

* The referencing field can be set to null (or) to the default value for the domain (by using set default)

SQL data types and schemas

Date and Time types in SQL

date : A date containing a (4 digit) year, month, day (ex) '2001-04-25'

time : The time of day in hours, minutes, seconds (ex) '09:30:00' (or) time(p) can be used to specify the number of fractional digits for seconds.

time with timezone : To store time-zone information along with the time.

timestamp : A combination of date and time. (ex) '2001-04-25 09:30:01.22' (or) timestamp(p) can be used to specify the number of fractional digits for seconds (default is 6)

timestamp with timezone : To store time-zone information along with timestamp.

interval : If x and y are of type date, then x-y is an interval whose value is the number of days from date x to date y.

SQL functions for date and time

current_date — returns the current date

current_time — returns the current time (with time zone)

localtime — returns the current time (without time zone)

current_timestamp — (with timezone)

localtimestamp — (local date and time without timezone)

Default values

```
create table student (ID varchar(5) primary key,
name varchar(20) not null, credits number(2)
                                default 0);
```

If no value is provided for the credits attribute, the value is set to 0.

(ex) `insert into student (ID, name) values ('IT01', 'anbu');`

Index creation : An index on an attribute of a relation is a data structure that allows the database system to find those tuples in the relation that have a specified value for that attribute efficiently, without scanning through all the tuples of the relation.

(ex) find the student detail of ID 22201

If index is created on attribute ID of relation student, the database system can find the record with any specified ID value such as 22201 or 44553, directly, without reading all the tuples of the student relation.

(ex) `Create index studentidindex on student(ID);`

When a user gives an SQL Query, the SQL Query processor uses the index, so the required tuple is quickly found without reading the whole relation.

Large - Object types : Many database applications need to store attributes that can be large (of the order of many kilobytes) such as photos, (or) very large (megabytes (or) gigabytes) such as video clips.

SQL provides Large object data types for Character data (clob) and binary data (blob).

(ex)
```
bookcontent  clob(10KB)
image        blob(10MB)
movie        blob(2GB)
```

User-defined types: Two forms of user-defined data types

- distinct types
- structured data types (oodbms)

create type clause is used to define new types.

(ex)
```
create type dollars as number(12,2) final;
create type pounds as number(12,2) final;
```

final keyword – optional for some databases
mandatory for SQL:1999

Use user define data type, to create a table.

(ex)
```
create table department (deptname varchar(20),
building varchar(15), budget dollars);
```

SQL had a similar but subtly different notion of domain which can add integrity constraint to an underlying type.

(ex)
```
create domain ddollars as numeric(12,2) not null;
```

Differences between types and domains

1. Domains can have constraints
2. Domains are not strongly typed. (ie) values of one domain type can be assigned to values of another domain type as long as underlying types are compatible.

domain with check clause

```
Create domain salary number(10,2) constraint
salarytest check(value >= 29000.00);
```

Authorization : It include

* Authorization to read data
* Authorization to insert new data
* Authorization to update data
* Authorization to delete data

} Each type of authorization is called Privilege

Authorization is done for all user, single user, none (or) combination of users on specified parts of a database, such as relation (or) view.

* When a user submits a query, SQL first checks if the query is authorized, based on the authorization that the user has been granted.
* If the query is not authorized, it is rejected.
* A user who has authorization may be allowed to pass on (grant) this authorization to other users. Similarly to withdraw (revoke) an authorization that was granted earlier.

Granting and Revoking of Privileges

The SQL allows the privileges select, insert, update and delete. The privilege all privileges can be used as a short form for all privileges.

Syntax:

```
grant <previlege list>
  on <relation (or) view name>
  to <user/role list>;
```

Syntax:

```
revoke <previlege list>
  on <relation (or) view name>
  from <user/role list>;
```

(ex) grant select on department to Amit, geetha;

grant update (budget) on department to Amit;

grant insert (bid, bname, budget) on department to Amit;

grant delete on department to Amit, geetha;

grant select on department to public;

* Public refers to all current and future users.

revoke select on department from Amit, geetha;

revoke update (budget) on department from Amit;

Roles: A set of roles is created in the database. Authorizations can be granted to roles.

Syntax for creating a Role: create role <role name>;

(ex): create role faculty;
create role hod;

Roles can then be granted privileges like users.

grant select on department to faculty;
grant select on department to hod;

Roles can be granted to users.

grant faculty to Amit;
grant hod to geetha;
grant faculty to hod;

So, the privileges of a user (or) role consists of

* All privileges directly granted to the user/role.
* All privileges granted to roles that have been granted to the user/role.

Authorization on views!

Create view ITfaculty as (select * from faculty where deptname = 'IT';

grant select on ITfaculty to Amit, Geetha;

Authorization on schema

* Only the owner of the schema can do any modification to the schema, such as creating (or) deleting relations, adding (or) dropping attributes and adding (or) dropping indices.

SQL has references privilege — It permits a user to declare foreign keys when creating relations.

(ex) grant references (deptname) on department to Amit;

(ie) Amit can create relations that reference the key deptname of department as foreignkey.

Transfer of privileges : A user who has been granted some form of authorization may be allowed to pass on this authorization to other users.

grant select on department to Amit with grant option;

Revoking of Privileges : A revocation of a privilege from a user may cause other users also to lose that privilege. This is called cascading revocation. To prevent cascading revocation, restrict is used.

(ex)
revoke select on department from Amit restrict;
The system returns error if any cascading revocation.