DBMS

Database: Database is a collection of inter-related data which helps in efficient retrieval, insertion and deletion of data. It stores data in the form of tables, schemas, views etc.

Database Management System: The software which enables you to manage the database. For example: MySQL,Oracle.

DBMS allows us to do:

Data definition: It helps in creation, modification and deletion of definition that define the organisation of data in the database.
Data Updation: It helps in insertion, modification and deletion of actual data in the database.
Data Retrieval: It helps in retrieval of data from the database which can be used by other applications.
User Administration: It helps in registering and monitoring users, enforcing data security, maintaining performance, maintaining data integrity etc.

Problems DBMS solves: (Disadvantages of file-based system)

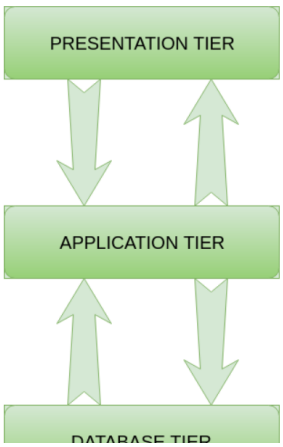
1. Redundancy of Data
2. Inconsistency of Data
3. Unauthorised Access
4. Difficult data Access
5. No Concurrent Access
6. No backup or recovery

RDBMS: Relational database means the data is stored as well as retrieved in the form of relations(tables)

SQL: Structured query language is the standard database language in RDBMS.

- Case insensitive
- Single line comments use double hyphen(--)
- Examples: MySQL,Oracle,SQL Server, PostgreSQL etc

DBMS 3-tier Architecture: divides the completes system into 3 inter-related but independent modules



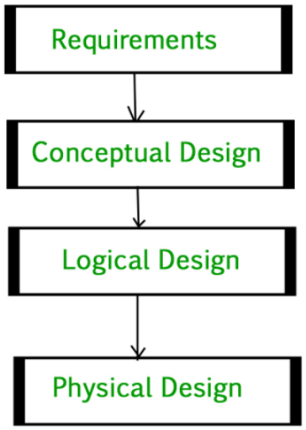
Advantages:
1. Enhanced Scalability (due to distributed deployment of application server. Now individual client server connection not required)
2. Data Integrity is maintained (because middle is there so data corruption can be avoided)
3. Security is improved (no direct interaction)

Disadvantages: Increase complexity of implementation and communication.

Data Independence: It means a change of data at one level should not affect another level. Two types:

1. Physical: Any change in location of tables and indexes doesn't affect the conceptual level or external view of data. (Easy to achieve)
2. Conceptual: Any change in conceptual schema should not affect external level schema. (Difficult to achieve) Like adding or deleting attributes of a table should not affect user's view but that's difficult because it usually does

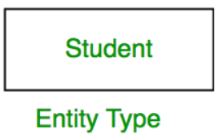
Phases of database design:



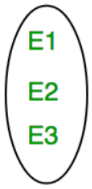
Conceptual: ER Model, Logical: Convert ER to relational

Entity: An object with physical existence (like car, person, house) or conceptual existence (like company, job, university course)

An entity is an object of **Entity Type** and set of all entities is called **Entity Set**.



Entity Type



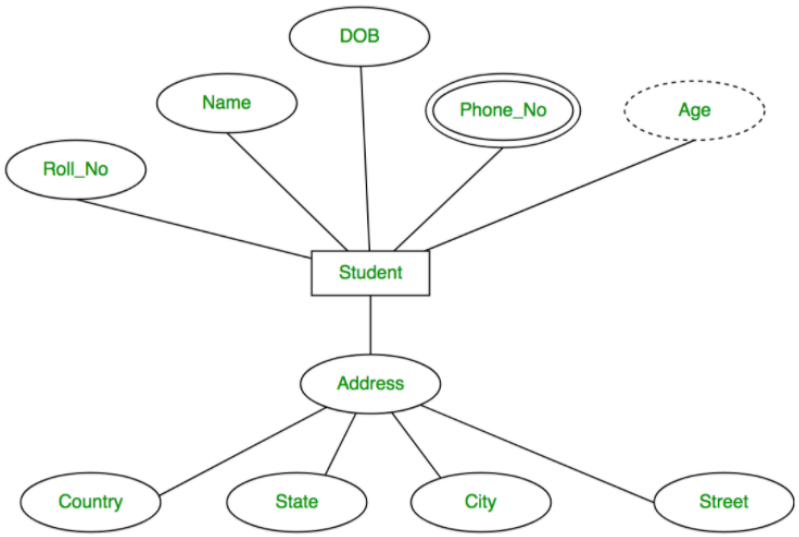
Entity Set

E1,E2,E3 are entities of type student.

Attributes: Properties which define the entity type
Represented by oval in ER model. 4 types:

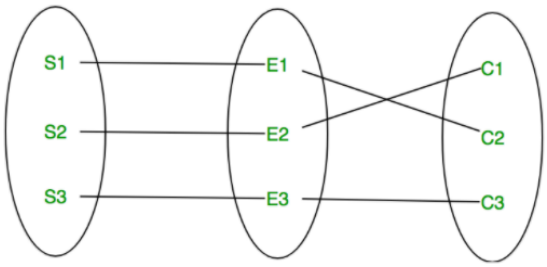
1. Key attribute:
    1. attribute which uniquely identifies each entity.
    2. Represented with oval and underline
    3. Example: Roll no.
2. Composite attribute:
    1. composed of many other attributes
    2. Represented with ovals connected to an oval
    3. Example:Address contains street, city, state etc
3. Multivalued attribute:
    1. attribute having more than one value
    2. Represented by double oval
    3. Example: A person can have multiple phone numbers
4. Derived attribute:
    1. can be derived from other attributes
    2. Represented by dashed oval
    3. Example: Age can be derived from DOB

Complete entity type student with its attributes:

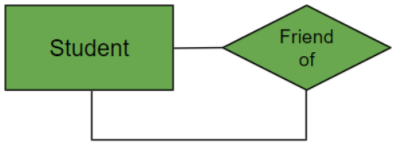


Relationship type represents the association between entity types.
Example: 'Enrolled in' is the relationship type

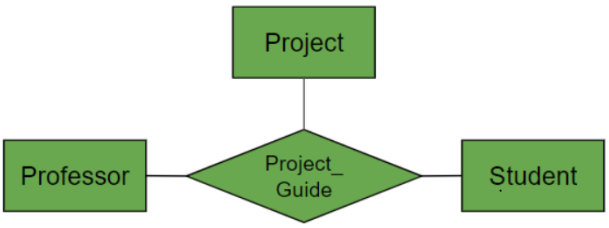
Relationship set: A set of relationships of the same type
Example:



Degree of a relationship set: Number of entity sets participating in a relationship set. 3 types:

1. Unary relationship:
    1. 

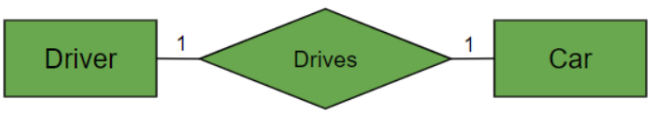
    2. A person is married to a person
    3. An employee manages an employee
2. Binary relationship:
    1. 

    2. Supplier supplies item
    3. Professor teaches subject
3. n-ary relationship:
    1. 

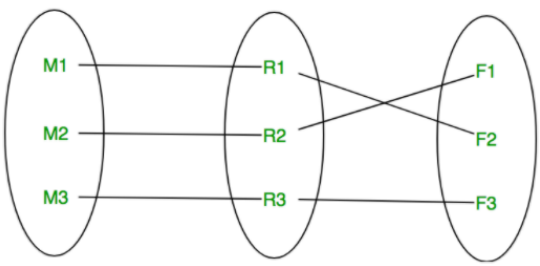
Cardinality: Number of times entity of an entity set participates in a relationship set

Three types:

1. One to one: one person hase one aadhar card and one aadhar card can belong to only one person

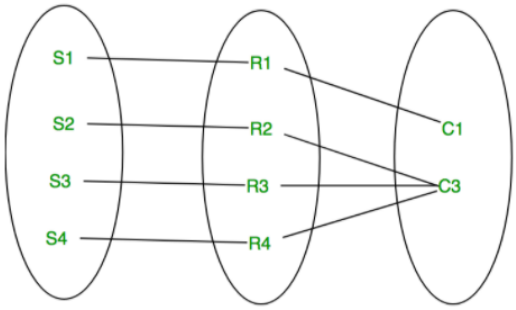


Set Representation:



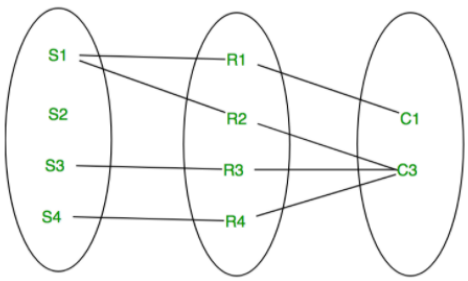
2. One to Many: Many employees work for 1 department

Set Representation:



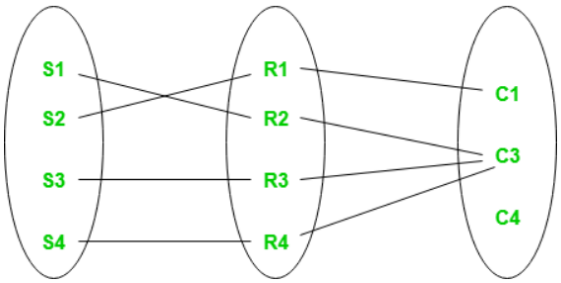
3. Many to many: Any number of customers can order any number of products

Set Representation:



Participation constraint: 2 types

1. Total participation: Each entity must participate
    1. Each employee must join a department
    2. 

2. Partial participation: Each entity may or may not participate
    1. Like in above example, some course might be there where no one enrolls so that's fine because of partial participation
    2. 

Weak Entity Type: Entity types for which key attribute can't be defined.

1. Represented by double rectangle.
2. Always total participation

Identifying Relationship: relationship between weak entity type and its identifying strong entity type. Represented by double diamond

Relational Model: A relational database stores data in the form of relations
Relation Schema: A relation schema represents name of the relation with its attributes. e.g.; STUDENT (ROLL_NO, NAME, ADDRESS, PHONE and AGE) is relation schema for STUDENT. If a schema has more than 1 relation, it is called Relational Schema.
Tuple: Each row in a relation
Relation Instance: Set of tuples in a relation at a particular instant
Degree: Number of attributes(columns)
Cardinality: Number of tuples
Column: Set of values for a particular attribute

Types of Constraints:

1. Domain Constraints: An attribute can only take values inside a domain range. Example: age>0
2. Key Integrity: Every relation should have at least one set of attributes that defines a tuple uniquely. This is called a key. It should be unique and can't have NULL values.
3. Referential Integrity: When one attribute of a relation can only take values from other attribute of same relation or any other relation

Anomalies: Irregularities
3 types: Insert, update and delete

1. Insertion Anomaly in Referencing Relation: We can't insert a row in referencing relation if the referencing attribute's value is not present in the referenced attribute.
2. Deletion/ Updation Anomaly in Referenced Relation: We can’t delete or update a row from REFERENCED RELATION if the value of REFERENCED ATTRIBUTE is used in the value of REFERENCING ATTRIBUTE

How to correct it? Use cascade

1. ON DELETE CASCADE: Deletes the tuples from referencing relation if the value is deleted from referenced relation
2. ON UPDATE CASCADE: Updates the tuples from referencing relation if the value is updated from referenced relation

---

Super key: Any set of attributes that allows to uniquely identify tuples
Candidate key: Proper subset of super keys
Composite key: Combination of two or more attributes being used as a primary key
Primary key: Most suitable candidate key
Alternate key: All candidate keys that are not primary key
Foreign key: This is a field in a table which uniquely identifies each row of another table

Armstrong Axioms: Used to verify whether an attribute is a candidate key or not

1. Axiom of reflexivity (A->A)
2. Axiom of transitivity (if A->B and B->C, then A->C)
3. Axiom of Augmentation(if A->B, then AC->BC)

So basically A->B means A derives B. So to find candidate key we have to make sure the selected key can derive all other attributes using the above axioms

Example: Given A -> B, A -> C, C -> D. Find candidate key
A->A (Axiom of reflexivity)
A->D (Axiom of transitivity)
A->B,A->C (given)
A^+=[ABCD] (closure of A)
So A is a candidate key

Database Normalization: The process of organizing attributes to reduce or eliminate data redundancy

Functional Dependency: A function dependency A → B means for all instances of a particular value of A, there is the same value of B. It is denoted by an arrow (→)
Example: {student_id,time}->{lecture_room}

Why do we need functional dependency? To carve out a few attributes to another table to reduce data redundancy. Example:

Student_ID	Name	Dept_Id	Dept_Name
101	ABC	10	CS
102	BCD	11	ECE
103	ABC	10	CS
104	XYZ	11	ECE
105	CDE	10	CS

From this table we can take out dept_id and dept_name to another table as dept_id->dept_name

Dept_Id	Dept_Name
10	CS
11	ECE

Types of functional dependencies:

1. Trivial: X->Y is trivial only when Y is a subset of X. Example: ABC->AB
2. Non Trivial: when Y is not a subset of X. Example: Id->name
3. Semi Non Trivial: when intersection of X and Y is not null. Example: AB->BC

Normal forms: used to eliminate or reduce redundancy

Types:

1. **First Normal Form:** A relation is in first normal form if every attribute in that relation is singled valued attribute.

Example: This table violates principle of first normal form and hence must be reduced. There are several methods:

Customer_ID	Name	Mob No
101	ABC	8860033933, 9899868189
102	BCD	8960133933
103	XYZ	8681899900, 9681888800
104	PQR	8189399888

Method 1: Involves creation of multiple columns like mob no 1,mob no 2 etc. But the problem is some places are left blank(wastage of space) because not everyone has 2 phone numbers

Method 2: Involves creation of multiple rows like create a new row with info (101,ABC,9899868189). But there is one problem that it duplicates customer_id which must be unique

Method 3: Involves creation of another table and shifting the repeated attributes to that table and linking it with previous table using customer_id

2. **Second Normal Form:** It should be 1NF and no partial dependency i.e. no non-prime attribute is dependent on any proper subset of any candidate key

So basically what partial dependency is that since the primary key is user_id+course_id, if some other column is dependent on only one of them, say course_id, then it is partially dependent.

User_ID	Course_ID	Course_Fee
1	CS101	5000
2	CS102	2000
1	CS102	2000
3	CS101	5000
4	CS102	2000
2	CS103	7000

In the above table, the candidate key is the combination of (User_ID, Course_ID) . The non-prime attribute is Course_Fee. Since this non-prime attribute depends partially on the candidate key, this table is not in 2NF.

To convert this into 2NF, one needs to split the table into two and creating a common link between two.

User_ID	Course_ID

Course_ID	Course_Fee

3. **Third Normal Form:** It should be in 2NF and no non-prime attribute must define another non-prime attribute(or you can say Non-prime attributes are not transitively dependent on prime attributes.)

Stud_No	Stud_Name	Stud_State	Stud_Country
101	Ram	Haryana	India
102	Ramesh	Punjab	India
103	Suresh	Punjab	India

In the above table Stud_no is the primary key but the dependencies are:
Stud_No → [Stud_Name, Stud_State, State_Country]
Student_State → Stud_Country
So student_state being a non-prime attribute is defining another non-prime attribute. Therefore not in 3NF form.
Problem is fixed by making two tables:
T1: Stud_No, Stud_Name, Stud_State
T2: Stud_State, Stud_Country

4. **Boyce-Codd Normal Form (BCNF):** Should be in 3NF and in every non-trivial functional dependency X --> Y, X should be a super key.

In simple words, no prime or non-prime attribute must define any prime attributes

STUD_ID	Subject	Prof_id
1001	DBMS	103
1001	OS	110
1002	DBMS	111

Functional Dependencies: (Stud_ID, Subject) → Prof_id, Prof_id → Subject
Candidate Keys: (Stud_ID, Subject), Prof_id, (Prof_id, Stud_ID)
Prof_id being a prime attribute defines another prime attribute Subject. So not in 3NF
To fix it, divide into two tables T1: STUD_ID,Prof_id, T2: Prof_id,Subject

5. 4th Normal form: It should be in BCNF and have no multivalued dependency

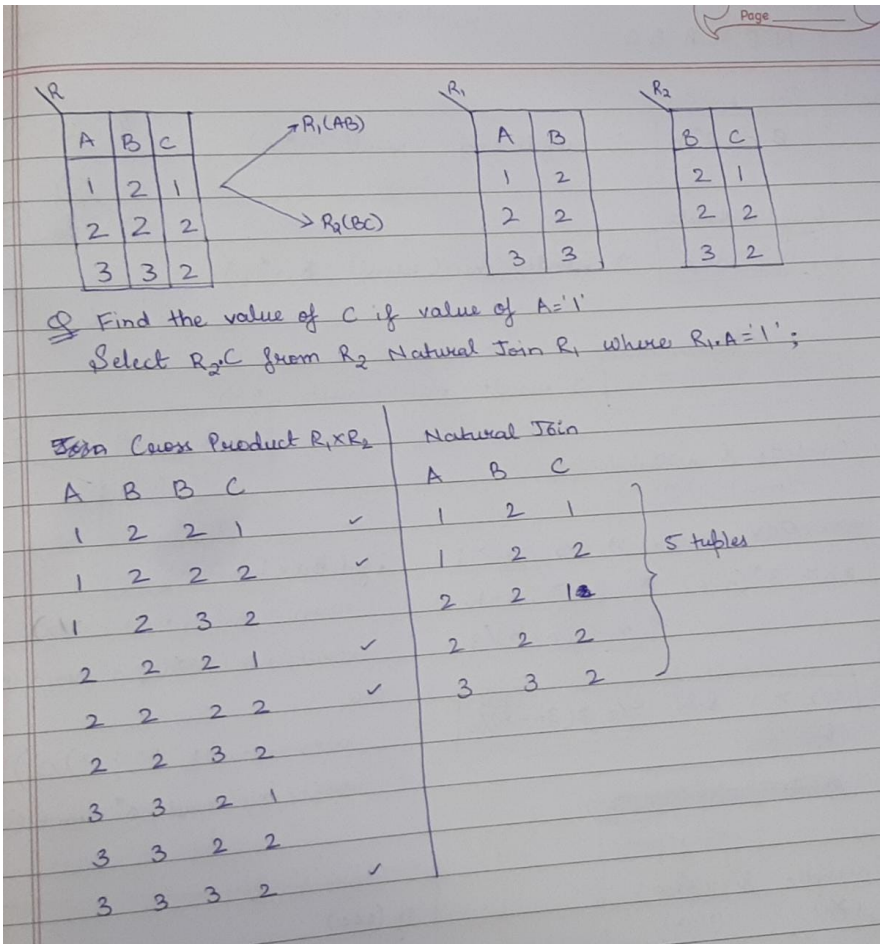
Example: Varun has 3 phone numbers and 3 emails so there will be total 9 rows (like (Varun, M1, E1), (Varun, M1, E2)..) . So to remove this just make separate table for mobile number and email

6. 5th Normal form: It should be in 4nf and have lossless decomposition which means decompose karke join karne pe same rehna chahiye

Lossless and Lossy Decomposition

When we do normalisation, we need to decompose table

Lossy Decomposition:



As we see here after decomposing and again joining we get a table of 5 tuples instead of the original 3. This is data inconsistency. The extra tuples are called spurious tuples.

So to prevent this we do lossless decomposition which has 3 conditions:

1. R1 U R2 = R
2. R1 ∩ R2 != Φ
3. The common attribute should be a candidate or super key in either R1 or R2 or both.

With these conditions we will select A as the common attribute instead of B in the above table. In the data will be consistent and is called lossless.

Joins: Join = Cross Product + Condition(select statement)

Types:

1. Natural Join: operation that creates an implicit join clause for you based on the common columns in the two tables being joined. Common columns are columns that have the same name in both tables.

Example: Select E_Name from Employees ,Department where Employees.Eid =Department.Eid; this can be written as:

Select E_Name from Employees Natural Join Department;

2. Self Join: When same table is made alias and multiplied by itself.

Select T1.S_id from Study as T1 , Study as T2 where T1.S_id = T2.S_id and T1.C_id <>T2.C_id
(<> means not equal to)

3. Equi Join: combines tables based on matching values in specified columns. Please remember that: The column names do not need to be the same. The resultant table contains repeated columns. It is possible to perform an equi join on more than two tables

Select E_Name from Emp,Dept where
Emp.E_no = Dept.E_no and Emp.Address = Dept.Location

4. Left Outer Join: Natural Join + Left Table

It gives the matching rows and the rows which are in left table but not in right table
Select emp_no,e_name,d_name,loc from emp left outer join dept on(emp.dept_no = dept.dept_no)
In this whatever is present in the left table comes in the main table if not present in the right table it is values of that are placed as NULL. But Left column k corresponding values to rahegi.

5. Right Outer Join: Natural Join+Right Table

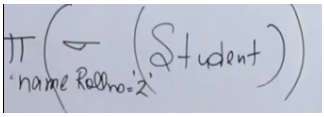
Right Rows whichever are there will be present. And corresponding value which are not present on the left table will be set NULL.
Select emp_no,e_name,d_name,LOC from emp Right Outer Join dept on (emp.dept_no = dept.dept_no)

Full Outer Join: Right Outer Join Union Left Outer Join

Relational Algebra: Procedural Query Language or Formal Query Language.

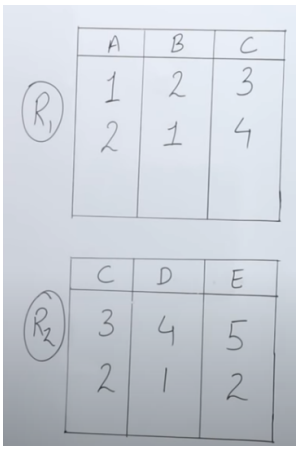
Projection means choosing which columns (or expressions) the query shall return. Always gives unique data only
Selection means which rows are to be returned.

Q: Retrieve the name of student whose roll no. is 2



Always remember projection is done at the end because we select what column we want to display at the end. If we do it earlier it might remove some column which is need for some function.

Cross Product: It is very important because joins can't happen without cross product. Also at least common column should be present
No. of columns in cross product=m+n
No. of rows in cross product=x*y



Set Difference: A-B= A but not B =A intersection B'
No. of columns must be same.
Domain of every column should be same
A-B!=B-A

Union: AUB
No of columns must be same
Domain of every column should be same

Division: It is a derived operator
A(X,Y)/B(Y) = return x values where for every value of y there is a tuple <x,y>
Q: Retrieve SId of students who are enrolled in **every/all** course

E(SId,CId)/C(CId)=S1

Relational Calculus: two types

1. Tuple Relational Calculus
2. Domain Relational Calculus

TRC: It is a non-procedural query language unlike relational algebra. Basically it tells you what to do and not how to do.

Query is expressed as
{ t | P(t) }
where t= resulting tuples, P(t) is known as predicate which is the conditions that are used to fetch t.
Basically jo jo P(t) ke conditions ko follow karega woh saare tuples output aayenge

Operations:

1. OR (V)
2. AND (∧)
3. NOT (¬)

Quantifiers:

1. There exists (∃) : ∃ t ∈ r (Q(t)) means there exists a tuple in t in relation r such that Q(t) is true
2. For all (∀) : ∀ t ∈ r (Q(t)) means Q(t) is true for all tuples in relation r

So the only thing you need to be careful of in TRC is unsafe expression like { s.name | ¬ Supplier(s) } which means all the supplier names which are not in supplier table which is an infinite set. Apart from that both TRC and RA (Relational Algebra) have same expressive power

Examples:

1. Find the loan number, branch, amount of loans of greater than or equal to 10000 amount.

```
{t| t ∈ loan  ∧ t[amount]>=10000}
```

2. Find the loan number for each loan of an amount greater or equal to 10000.

```
{t| ∃ s ∈ loan(t[loan number] = s[loan number]
                  ∧ s[amount]>=10000)}
```

https://www.geeksforgeeks.org/tuple-relational-calculus-trc-in-dbms/

Domain Relational Calculus: It is a non-procedural query language. It is expressed as

```
{ < x1, x2, x3, ..., xn > | P (x1, x2, x3, ..., xn ) }
```

where P is predicate which can have comparison operators, connectives like and, or, not, and quantifiers

Examples:

1. Find the loan number, branch, amount of loans of greater than or equal to 100 amount.

```
{<l, b, a> | <l, b, a> ∈ loan ∧ (a ≥ 100)}
```

2. Find the names of all customers having a loan at the “Main” branch and find the loan amount .

```
{<c, a> | ∃ l (<c, l> ∈ borrower ∧ ∃ b (<l, b, a> ∈ loan ∧ (b = “Main”)))}
```

https://www.geeksforgeeks.org/domain-relational-calculus-in-dbms/

Procedural SQL (PL-SQL): Not only tells us what to do but also how to do. It has a block of code.
It contains:

1. Functions
2. Procedure
3. Trigger
4. Cursor

Has 3 parts:

1. Declaration: first variable name then variable type (Ex: a int)
2. Executable Code: starts with begin ends with end. To assign value we use := (Example: a:=10)
3. Exception handling: any error (like divide by zero)

Transaction: It is a set of operations used to perform a logical unit of work. A transaction generally represents change in database. Two major operations are read and write.
Example: Someone wants to transfer INR 500 from one bank account (maintained by variable A) to another bank account (maintained by variable B)

At any instance of moment, there can be many such transactions T1,T2,T3...,Tn submitted by same or many users. If these transactions are executed sequentially:

1. waiting time will be more for some transactions than expected
2. Many CPU cycles will be idle when I/O operations are going on

Concurrency is achieved by DBMS which interleaves actions of various transactions. However simultaneously executing transactions may act on same variables in the database and so the order of execution must be satisfied by such concurrent execution. This is called controlled concurrency.

Properties of transaction:

1. Atomicity: All or nothing
2. Consistency: takes DB from one consistent state to another without necessarily preserving interim consistency
3. Isolation: Concealed form each other. The change of variable by one transaction are not realizable by another transaction until written back
4. Durability: Once a transaction commits, its updates survive in the database.

Lifecycle of a transaction:

Anomalies due to multiple transactions happening together (Interleaved Execution) :

1. Dirty Read problem: reading uncommitted data (WR conflicts)

In this example first t1 changes the value and writes it and then t2 reads it but after that t1 fails because of which the entire t1 is reversed (rollback) because of atomicity property of transactions. But t2 uses the changed value and completes the transaction.

2. Unrepeatable Reads: RW conflict

So t1 reads and then t2 completes the transaction and changes value. Then t1 reads again and gets a different value of the same variable without itself changing anything.

3. Lost Update problem: Overwriting Uncommitted Data (WW conflicts)

t1 reads and operates but before writing it t2 comes reads, operates and writes. After that when t1 writes it overwrites t2's variable value.

Scheduling Transactions:

1. Serial schedule: Does not interleave the actions of different transactions. Will always take database to consistent state.
2. Non-serial schedule: Interleaves the actions of different transactions. May take database to consistent state or may generate unwanted result.
3. Serializable schedule: A non-serial schedule equivalent to some serial execution of the transactions.
4. Complete Schedule: Commit or abort statement is used at the end of the transaction to permanently record the values of the data items in database.
5. Recoverable Schedule: In the dirty problem, after t1 rolls back we try to rollback t2 but since it is already committed we can't rollback. This is non-recoverable schedule. But if t2 commits only after the commit of the transaction from which it reads data, the schedule is recoverable.
6. Cascade-less schedule: It is a bit stricter condition. One transaction reads only those data already committed by another transaction. No need of cascading of rollback in this case. It is recoverable.
7. Strict schedules: More strict. One transaction reads or writes only those data already committed by another transaction. No need of cascading of rollback in this case. It is cascade-less.

8. Equivalent schedule: For any database state, the effect of executing the first schedule is identical to the effect of executing the second schedule. Two ways:
    1. Result Equivalent Schedules: All the schedules produce same database state for a given database state as input

2. Conflict Equivalent Schedules: All the schedules should have same conflicts and in same order. Conflicting actions are any combination of the following:
    1. r(A) by T_i and w(A) by T_j
    2. w(A) by T_i and r(A) by T_j
    3. w(A) by T_i and w(A) by T_j

Non conflicting are r(A) by T_i and r(A) by T_j or operation on two different data items

Example:

S1 and S2 have same conflicts in same order. Thus they are conflict equivalent.

If a non-serial schedule S1 is conflict equivalent to a serial schedule S2, then S1 is termed as conflict serializable.

Testing Conflict serializability:
Steps:

1. Start drawing a graph, which is called as precedence graph. Create a node Ti for each participating transaction in the schedule S under test.
2. For each case where Tj executes R(X) after Ti executing W(X), we draw an edge from Ti to Tj.
3. For each case where Tj executes W(X) after Ti executing R(X), we draw an edge from Ti to Tj.
4. For each case where Tj executes W(X) after Ti executing W(X), we draw an edge from Ti to Tj.
5. Cycle in precedence graph -> NOT conflict serializable
   NO cycle in precedence graph -> conflict serializable