RDBMS Concepts – Sivagami S

RDBMS Concepts

Agenda

1. RDBMS Introduction
2. RDBMS vs File System
3. RDBMS Concepts
4. Keys
5. Constraints
6. Data Models
7. E-R Model
8. Normalization
9. ACID Properties
10. Transaction Management
11. Concurrency Control

RDBMS Introduction

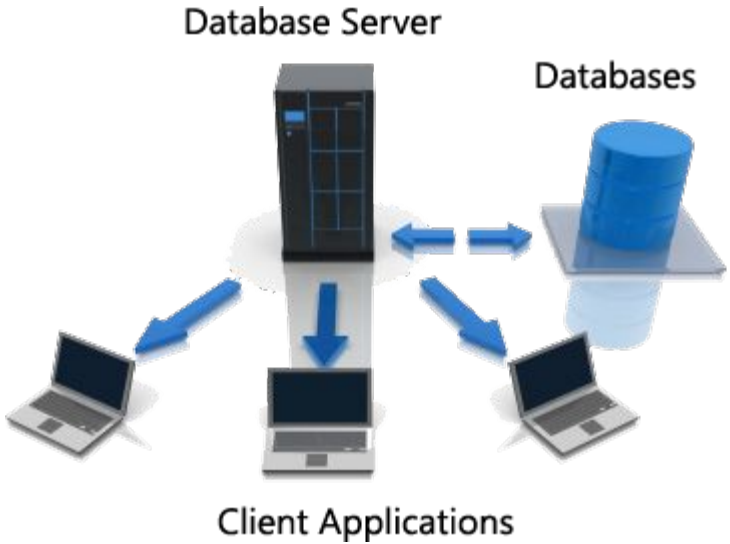
DBMS (Database management system) - A collection of programs that enables user to store, retrieve, update and delete information from a database.

RDBMS (Relational Database Management System) – Is a database management system (DBMS) that is based on the relational model.
In relational model, data is represented in terms of tuples(rows).

Popular RDBMS softwares.

What is database Server ?

- The term database server may refer to both hardware and software used to run a database.
- As software, a database server is the back-end portion of a database application.
- This back-end portion is sometimes called the instance.
- It may also refer to the physical computer used to host the database.



DBMS VS FILE SYSTEM

	File System	DBMS
1.	File system is a software that manages and organizes the files in a storage medium within a computer.	DBMS is a software for managing the database.
2.	Redundant data can be present in a file system.	In DBMS there is no redundant data.
3.	It doesn’t provide backup and recovery of data if it is lost.	It provides backup and recovery of data even if it is lost.
4.	There is no efficient query processing in file system.	Efficient query processing is there in DBMS.
5.	There is less data consistency in file system.	There is more data consistency because of the process of normalization.

DBMS VS FILE SYSTEM

	File System	DBMS
6.	It is less complex as compared to DBMS.	It has more complexity in handling as compared to file system.
7.	File systems provide less security in comparison to DBMS.	DBMS has more security mechanisms as compared to file system.
8.	It is less expensive than DBMS.	It has a comparatively higher cost than a file system.

RDBMS Concepts

SQL : Is a language designed specifically for communicating with databases. SQL is an ANSI (American National Standards Institute) standard.

What is Table ?

Collection of data elements organized in terms of rows and columns.
Most simplest form of data storage.
Convenient representation of relations.

What is a Record ?

A single entry in a table.
Represents set of related data.

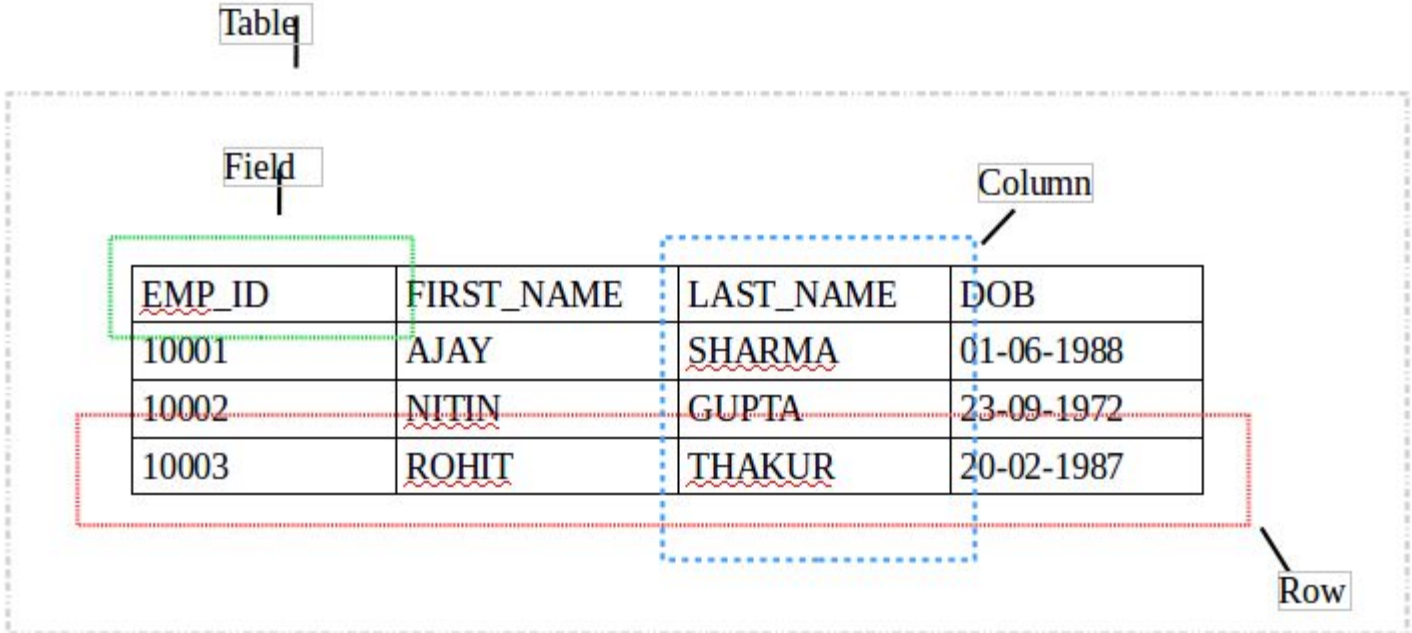
RDBMS Concepts

What is Field ?
Table consists of several records(row).
Each record can be broken into several smaller entities known as Fields.

What is a Column ?
Column is a set of value of a particular type.
The term Attribute is also used to represent a column.

What is a Schema ?
Schema refers to the organization of data as a blueprint of how the database is constructed

RDBMS Concepts



EMP_ID	FIRST_NAME	LAST_NAME	DOB
10001	AJAY	SHARMA	01-06-1988
10002	NITIN	GUPTA	23-09-1972
10003	ROHIT	THAKUR	20-02-1987

Database Keys

Used to establish and identify relation between tables.
Ensure that each record within a table can be uniquely identified.

Classification of Keys :
Super Key > Uniquely identifies each record within a table.
Superset of Candidate key.

Candidate Key > Set of fields from which primary key can be selected.
It is an attribute or set of attribute that can act as a primary key for a table.

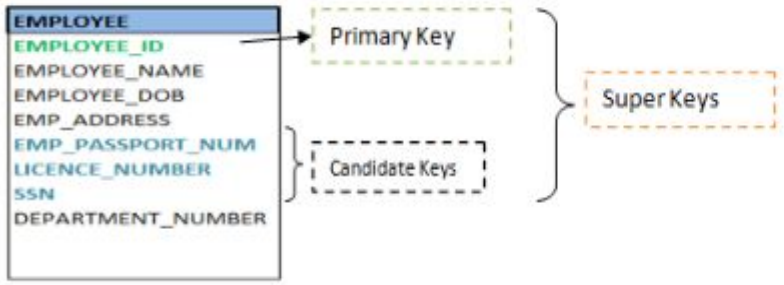
Primary Key > Is a candidate key that is most appropriate to become main key of the table.

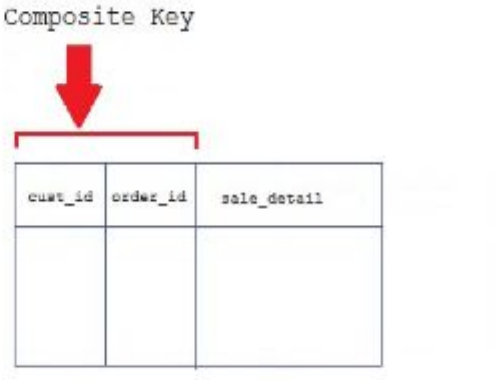
1. Uniquely identify each record in a table.
2. A table can have only one primary key.

Database Keys

Composite Key > Key that consist of two or more attributes that uniquely identify an entity occurrence.
Any attribute that makes up the Composite key is not a simple key in its own.

Secondary/Alternative key > Candidate key which are not selected for primary key are known as secondary/ Alternative keys.





Constraints

NOT NULL constraint
Ensures that column does not accept nulls

UNIQUE constraint
Ensures that all values in column are unique

DEFAULT constraint
Assigns value to attribute when a new row is added to table

CHECK constraint
Validates data when attribute value is entered

Data Modeling Concepts

Conceptual Data Model

A conceptual data model identifies the highest-level relationships between the different entities. Features of conceptual data model include:

1. Includes the important entities and the relationships among them.
2. No attribute is specified.
3. No primary key is specified.

Data Modeling Concepts

Logical Data Model

A logical data model describes the data in as much detail as possible, without regard to how they will be physical implemented in the database. Features of a logical data model include:

Includes all entities and relationships among them.

All attributes for each entity are specified.

The primary key for each entity is specified.

Foreign keys are specified.

Normalization occurs at this level.

The steps for designing the logical data model are as follows:

1. Specify primary keys for all entities.
2. Find the relationships between different entities.
3. Find all attributes for each entity.
4. Resolve many-to-many relationships.
5. Normalization.

Data Modeling Concepts

Physical Data Model

Physical data model represents how the model will be built in the database. A physical database model shows all table structures, including column name, column data type, column constraints, primary key, foreign key, and relationships between tables. Features of a physical data model include:

1. Specification all tables and columns.
2. Foreign keys are used to identify relationships between tables.
3. Denormalization may occur based on user requirements.
4. Physical considerations may cause the physical data model to be quite different from the logical data model.
5. Physical data model will be different for different RDBMS. For example, data type for a column may be different between MySQL and SQL Server.

Data Modeling Concepts

Physical Data Model Continuation..

The steps for physical data model design are as follows:

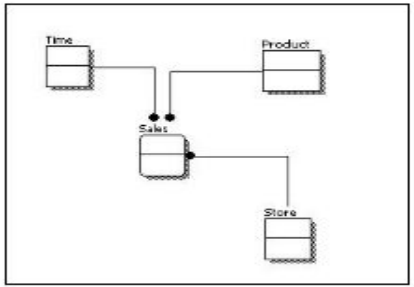
1. Convert entities into tables.
2. Convert relationships into foreign keys.
3. Convert attributes into columns.
4. Modify the physical data model based on physical constraints / requirements.

Data Modeling Concepts

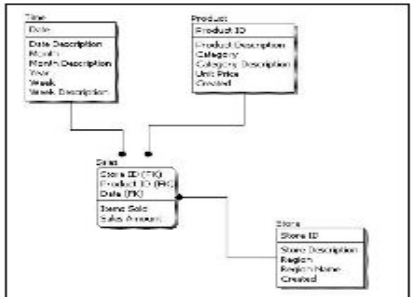
Feature	Conceptual	Logical	Physical
Entity Names	✓	✓	
Entity Relationships	✓	✓	
Attributes		✓	
Primary Keys		✓	✓
Foreign Keys		✓	✓
Table Names			✓
Column Names			✓
Column Data Types			✓

Below we show the conceptual, logical, and physical versions of a single data model.

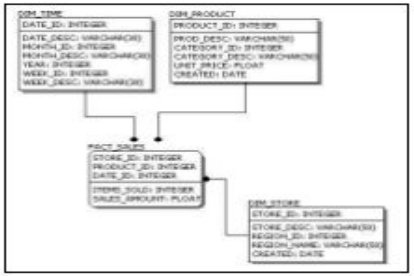
Conceptual Model Design



Logical Model Design



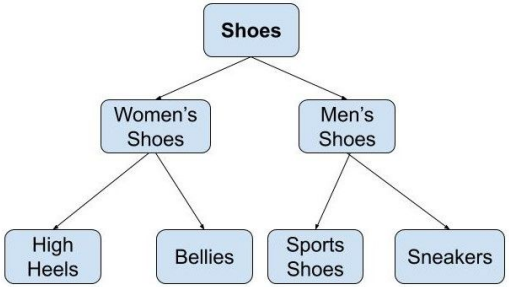
Physical Model Design



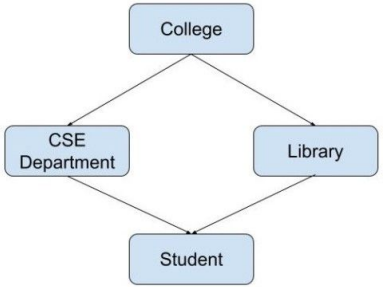
Data Models

Data Model gives us an idea that how the final system will look like after its complete implementation. It defines the data elements and the relationships between the data elements. Data Models are used to show how data is stored, connected, accessed and updated in the database management system. Here, we use a set of symbols and text to represent the information so that members of the organisation can communicate and understand it. Relational model is the most widely used model. Apart from the Relational model, there are many other types of data models

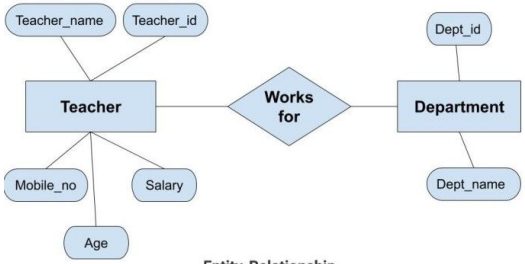
Types of Data Models



Hierarchical Model



Network Model

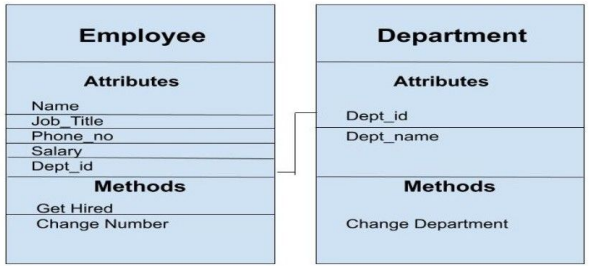


Entity-Relationship Model

Emp_id	Emp_name	Job_name	Salary	Mobile_no	Dep_id	Project_id
AfterA001	John	Engineer	100000	9111037890	2	99
AfterA002	Adam	Analyst	50000	9587569214	3	100
AfterA003	Kande	Manager	890000	7895212355	2	65

EMPLOYEE TABLE

Relational model



Object_Oriented_Model

Items

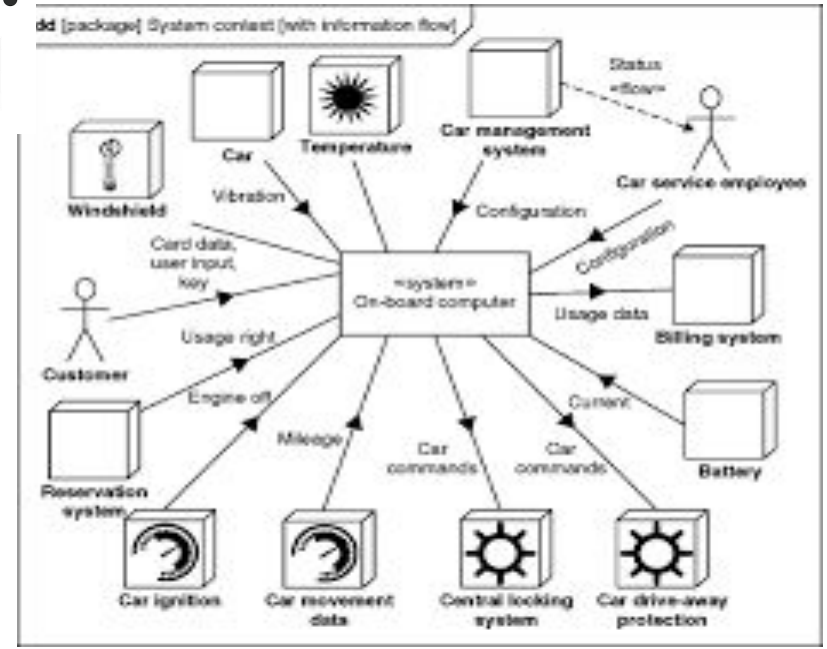
Identifiers	Name
89	The world cup
95	Is being hosted
40	By London
44	from
10	30 May 2020

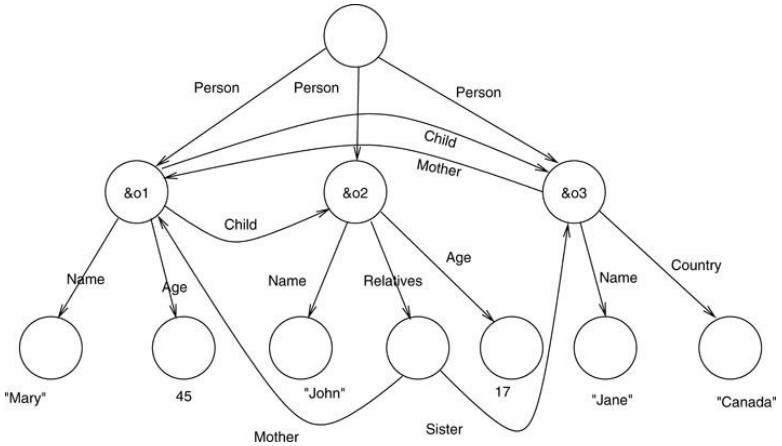
Links

Identifiers	Source	Verb	Target
70	89	95	40
75	70	44	10

ASSOCIATIVE MODEL

Types of Data Models





Semi Structured Model

Flat File Model

	Route No.	Miles	Activity
Record 1	I-95	12	Overlay
Record 2	I-495	05	Patching
Record 3	SR-301	33	Crack seal

Object-Oriented Model

Object 1: Maintenance Report

Date	
Activity Code	
Route No.	
Daily Production	
Equipment Hours	
Labor Hours	

Object 1 Instance

01-12-01
24
I-95
2.5
6.0
6.0

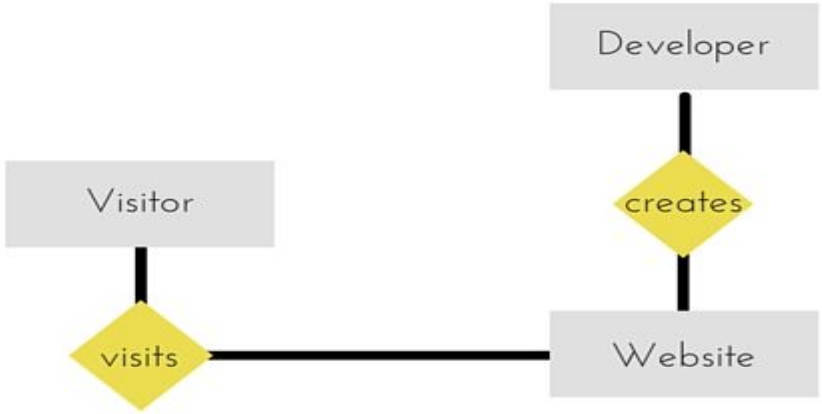
Object 2: Maintenance Activity

Activity Code	
Activity Name	
Production Unit	
Average Daily Production Rate	

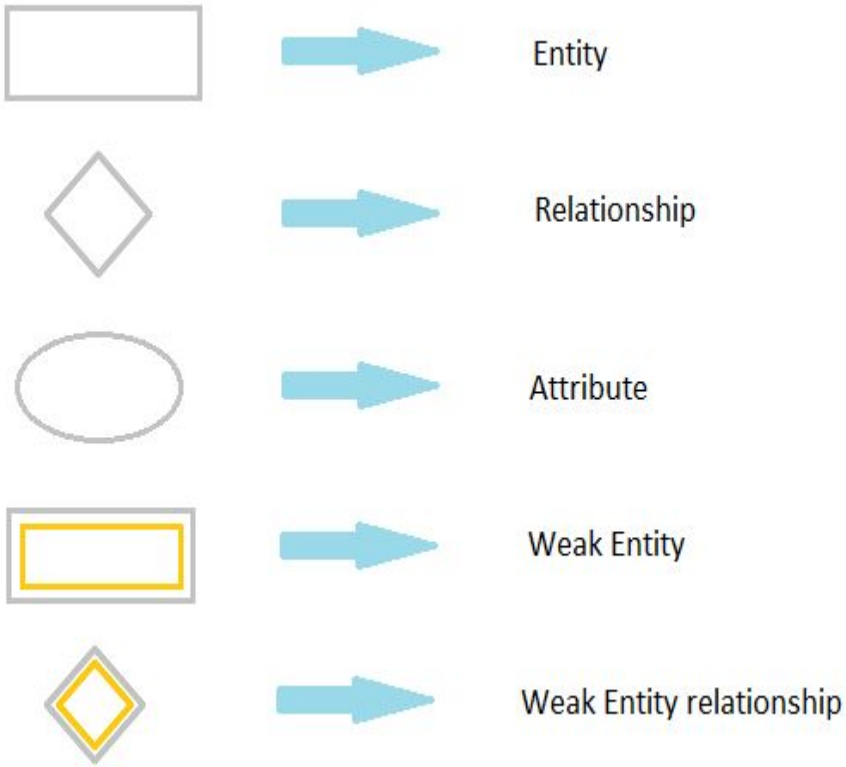
ER - Diagram

E-R Diagrams : Entity Relation Diagram

ER-Diagram is a visual representation of data that describes how data is related to each other.

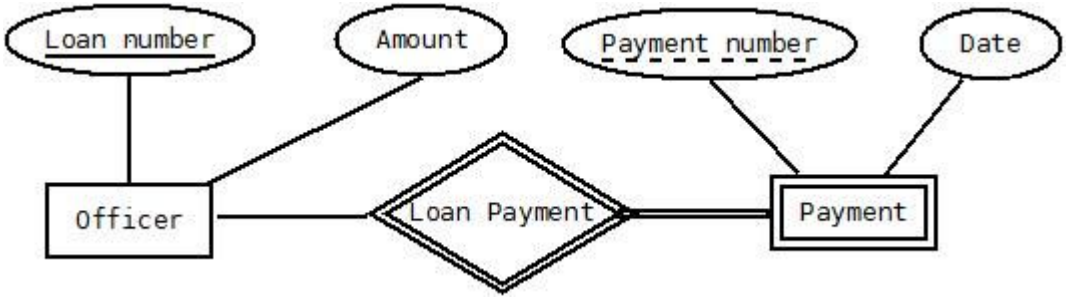


ER: Symbols and Notations



ER: Symbols and Notations

Derived attribute

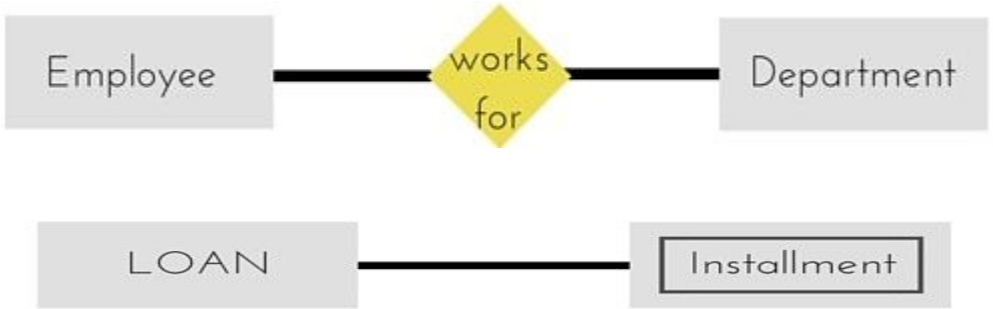


Key of a weak entity &
Total Participation
representation

ER Diagram Components

1)Entity : An Entity can be any object, place, person or class.
Represented using rectangles.

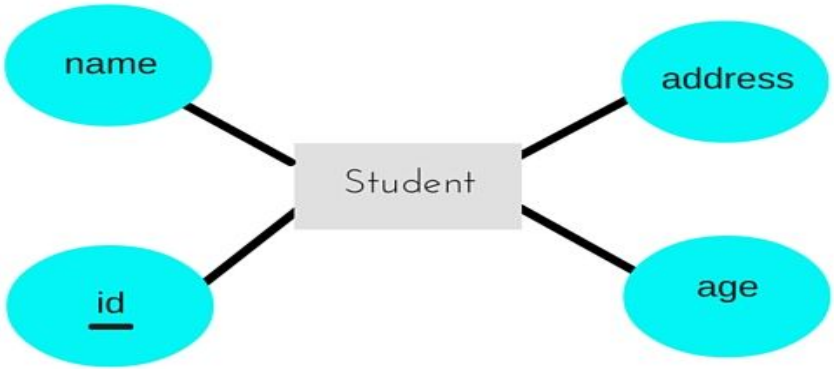
Weak Entity: Entity that depends on another entity.
Weak entity doesn't have key attribute of their own.
Double rectangle represents weak entity.



ER Diagram Components

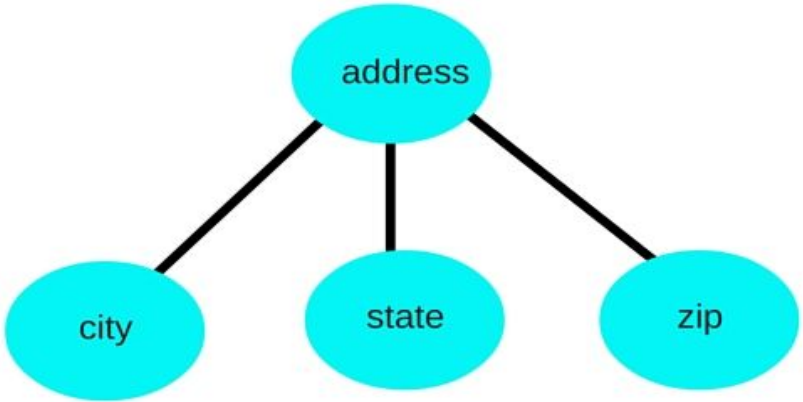
2) Attribute : An Attribute describes a property or characteristic of an entity.
An attribute is represented using eclipse.

Key Attribute > Represents the main characteristic of an Entity.
It is used to represent Primary key.
Ellipse with underlying lines represent Key Attribute.



ER Diagram Components

Composite Attribute > An attribute can also have their own attributes. These attributes are known as composite attribute



ER Diagram Components

3) Relationship : A Relationship describes relations between entities. Relationship is represented using diamonds.

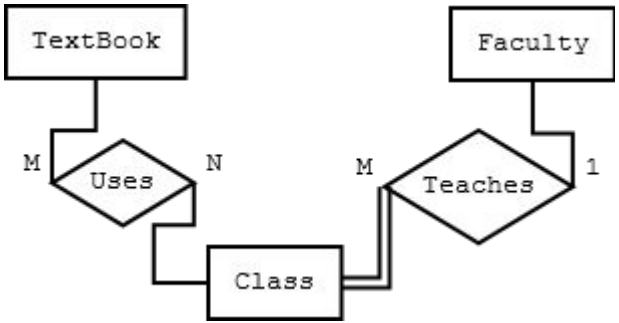
There are three types of relationship that exist between Entities.
3.1 > Binary Relationship : Binary Relationship means relation between two Entities.

This is further divided into three types.
3.1.1 > One to One : This type of relationship is rarely seen in real world.

ER Diagram Components

3.1.2 > One to Many : It reflects business rule that one entity is associated with many number of same entity.

3.1.3> Many to Many : It reflects business rule that many entities can be associated with many entity.



Normalization

1NF, 2NF, 3NF, and BCNF

Normalization is a process that “improves” a database design by generating relations that are of higher normal forms.

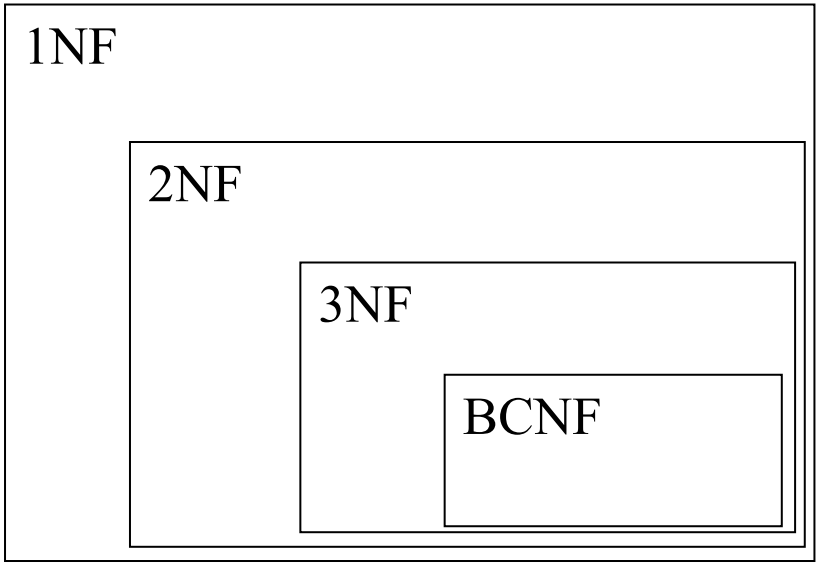
The *objective* of normalization:
“*to create relations where every dependency is on the key, the whole key, and nothing but the key*”.

Normalization

1NF is considered the weakest,
2NF is stronger than 1NF,
3NF is stronger than 2NF, and
BCNF is considered the strongest

Also,
any relation that is in BCNF, is in 3NF;
any relation in 3NF is in 2NF; and
any relation in 2NF is in 1NF.

Normalization



a relation in BCNF, is also in 3NF

a relation in 3NF is also in 2NF

a relation in 2NF is also in 1NF

Normalization

We consider a relation in BCNF to be fully normalized.

The benefit of higher normal forms is that update semantics for the affected data are simplified.

This means that applications required to maintain the database are simpler.

A design that has a lower normal form than another design has more redundancy. Uncontrolled redundancy can lead to data integrity problems.

First we introduce the concept of ***functional dependency***

Functional Dependencies

We say an attribute, B, has a *functional dependency* on another attribute, A, if for any two records, which have
the same value for A, then the values for B in these two records must be the same. We illustrate this as:

A $\Longrightarrow$ B

Example: Suppose we keep track of employee email addresses, and we only track one email address for each employee. Suppose each employee is identified by their unique employee number. We say there is a functional dependency of email address on employee number:

employee number $\Longrightarrow$ email address

Functional Dependencies

EmpNum	EmpEmail	EmpFname	EmpLname
123	jdoe@abc.com	John	Doe
456	psmith@abc.com	Peter	Smith
555	alee1@abc.com	Alan	Lee
633	pdoe@abc.com	Peter	Doe
787	alee2@abc.com	Alan	Lee

If EmpNum is the PK then the FDs:

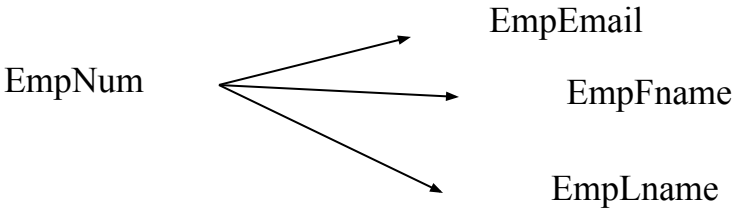
EmpNum $\Longrightarrow$ EmpEmail
EmpNum $\Longrightarrow$ EmpFname
EmpNum $\Longrightarrow$ EmpLname

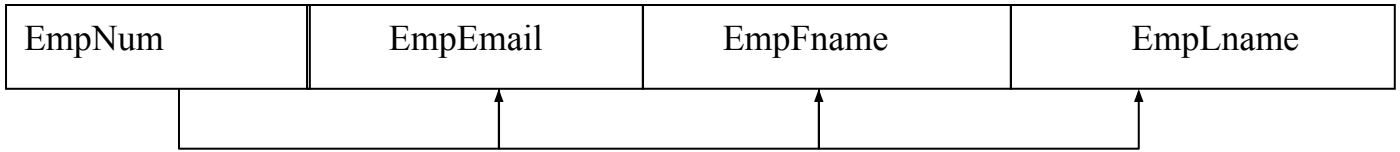
must exist.

Functional Dependencies

EmpNum ⟹ EmpEmail
EmpNum ⟹ EmpFname
EmpNum ⟹ EmpLname

3 different ways you might see FDs depicted





Determinant

Functional Dependency

EmpNum ⟹ EmpEmail

Attribute on the LHS is known as the ***determinant***

- EmpNum is a determinant of EmpEmail

Transitive dependency

Consider attributes A, B, and C, and where

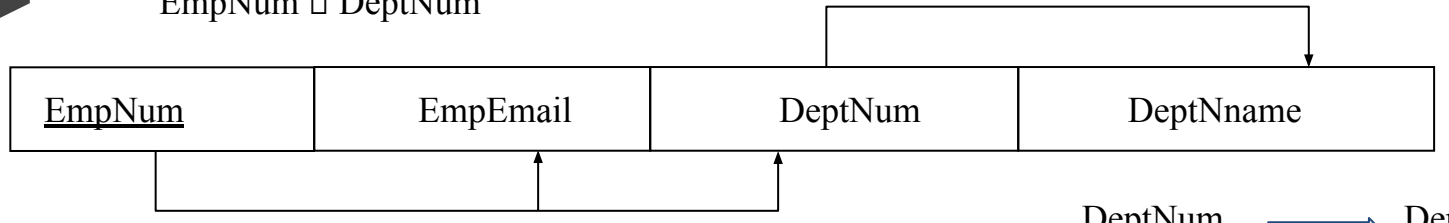
$A \Longrightarrow B$ and $B \Longrightarrow C$.

Functional dependencies are transitive, which means that we also have the functional dependency $A \Longrightarrow C$

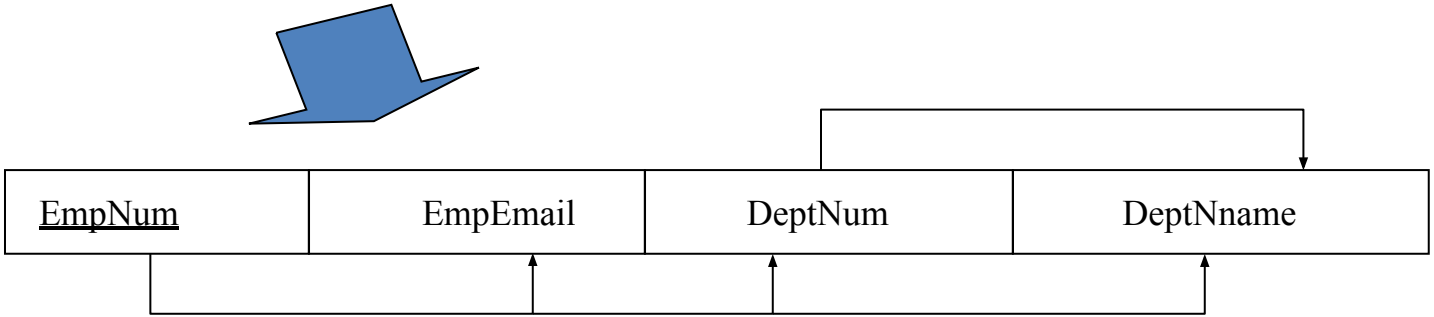
We say that C is transitively dependent on A through B.

Transitive dependency

EmpNum DeptNum



DeptNum ⟹ DeptName

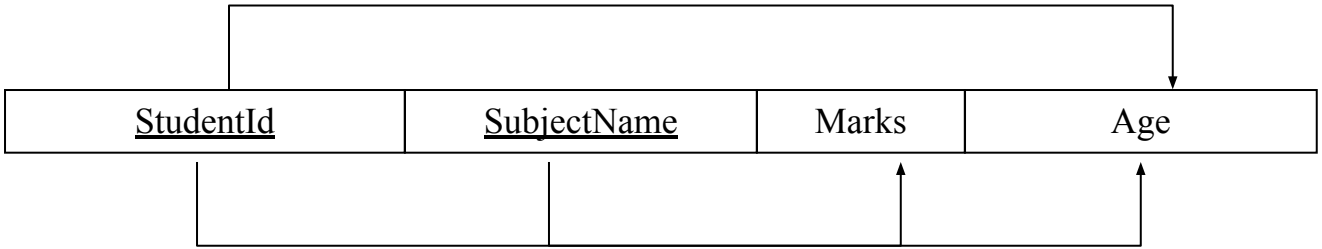


DeptName is *transitively dependent* on EmpNum via DeptNum

EmpNum ⟹ DeptName

Partial dependency

A **partial dependency** exists when an attribute B is functionally dependent on an attribute A, and A is a component of a multipart candidate key.



Candidate keys: {StudentId,SubjectName}
Age is *partially dependent* on {StudentId,SubjectName} as StudentId is a determinant of Age and StudentId is part of a candidate key

First Normal Form

First Normal Form

We say a relation is in **1NF** if all values stored in the relation are single-valued and atomic.

1NF places restrictions on the structure of relations. Values must be simple.

First Normal Form

The following in **not** in 1NF

EmpNum	EmpPhone	EmpDegrees
123	233-9876	
333	233-1231	BA, BSc, PhD
679	233-1231	BSc, MSc

EmpDegrees is a multi-valued field:

employee 679 has two degrees: *BSc* and *MSc*

employee 333 has three degrees: *BA, BSc, PhD*

First Normal Form

EmpNum	EmpPhone	EmpDegrees
123	233-9876	
333	233-1231	BA, BSc, PhD
679	233-1231	BSc, MSc

To obtain 1NF relations we must, without loss of information, replace the above with two relations - see next slide

First Normal Form

- **Employee**

EmpNum	EmpPhone
123	233-9876
333	233-1231
679	233-1231

- **EmployeeDegree**

EmpNum	EmpDegree
333	BA
333	BSc
333	PhD
679	BSc
679	MSc

- An outer join between Employee and EmployeeDegree will produce the information we saw before

Second Normal Form

- A relation is in **2NF** if it is in 1NF, and every non-key attribute is fully dependent on each candidate key. (That is, we don't have any partial functional dependency.)

- 2NF (and 3NF) both involve the concepts of key and non-key attributes.
- A *key attribute* is any attribute that is part of a key; any attribute that is not a key attribute, is a *non-key attribute*.
- Relations that are not in BCNF have data redundancies
- A relation in 2NF will not have any partial dependencies

Second Normal Form

- Consider this **InvLine** table (in 1NF)**:**

InvNum	LineNum	ProdNum	Qty	InvDate

- InvNum, LineNum ⟶ ProdNum, Qty
- InvNum, ProdNum ⟶ LineNum, Qty

- There are two candidate keys.
- Qty is the only non-key attribute, and it is dependent on InvNum

InvNum ⟶ InvDate

- Since there is a determinant that is not a candidate key, InvLine is **not BCNF**
- InvLine is **not 2NF** since there is a partial dependency of InvDate on InvNum

- InvLine is only in **1NF**

Second Normal Form

InvLine

• InvNum	• LineNum	• ProdNum	• Qty	• InvDate

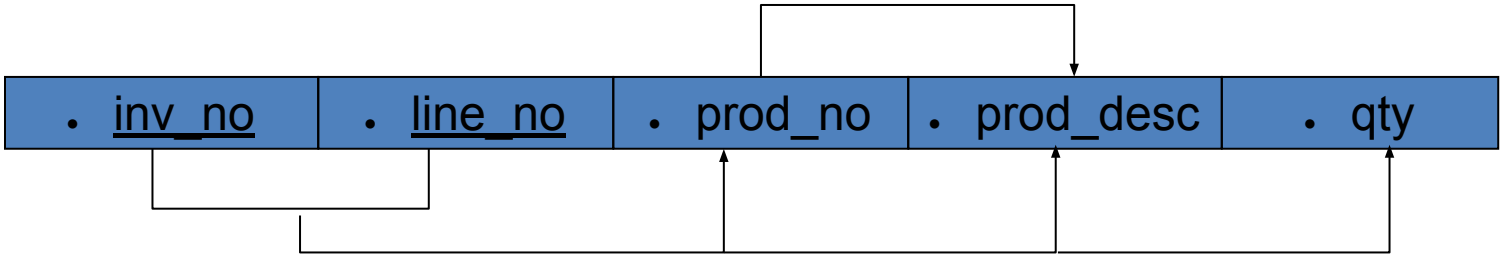
- The above relation has redundancies: the invoice date is repeated on each invoice line.
- We can *improve* the database by decomposing the relation into two relations:

• InvNum	• LineNum	• ProdNum	• Qty

• InvNum	• InvDate

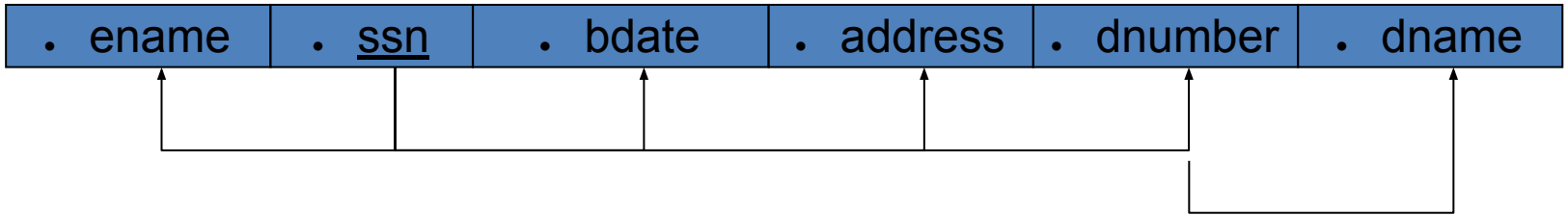
- Question: What is the highest normal form for these relations? 2NF? 3NF? BCNF?

- Is the following relation in 2NF?



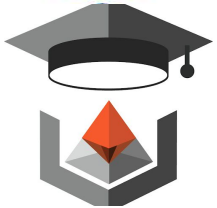
- 2NF, but not in 3NF, nor in BCNF:

- **EmployeeDept**



- since dnumber is not a candidate key and we have:

- dnumber → dname.



Third Normal Form

- A relation is in **3NF** if the relation is in 1NF and all determinants of *non-key* attributes are candidate keys
  That is, for any functional dependency: X → Y, where Y is a non-key attribute (or a set of non-key attributes), X is a candidate key.
- This definition of 3NF differs from BCNF only in the specification of non-key attributes - 3NF is weaker than BCNF. (BCNF requires all determinants to be candidate keys.)
- A relation in 3NF will not have any transitive dependencies
  of non-key attribute on a candidate key through another non-key attribute.

Boyce-Codd Normal Form

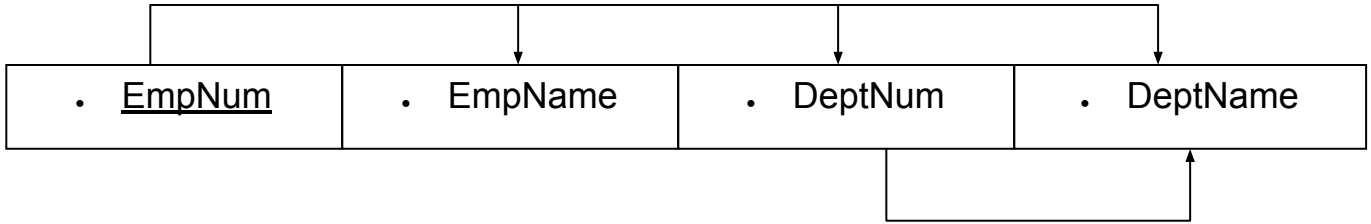
BCNF is defined very simply:

> a relation is in BCNF if it is in 1NF and if every determinant is a candidate key.

- If our database will be used for OLTP (on line transaction processing), then BCNF is our target. Usually, we meet this objective. However, we might denormalize (3NF, 2NF, or 1NF) for performance reasons.

Third Normal Form

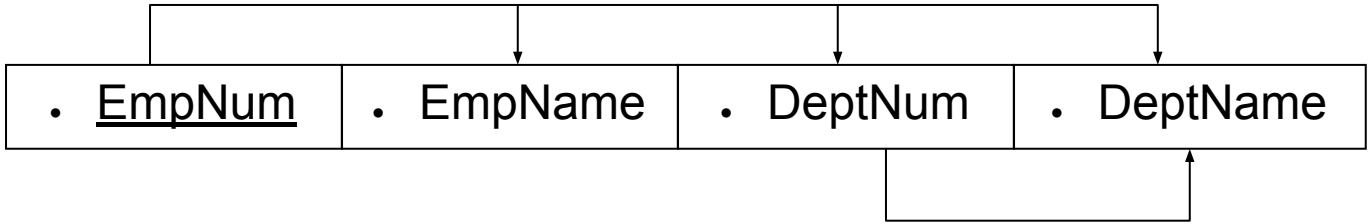
Consider this **Employee** relation



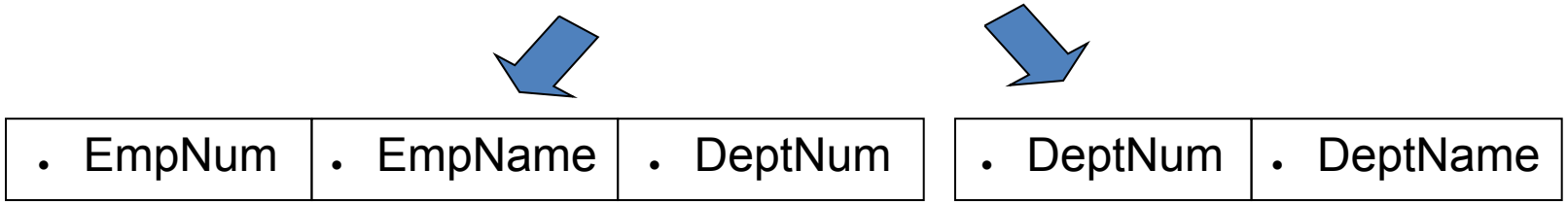
- EmpName, DeptNum, and DeptName are non-key attributes.
- DeptNum determines DeptName, a non-key attribute, and DeptNum is not a candidate key.

- Is the relation in 3NF? … no
- Is the relation in 2NF? … yes
- Is the relation in BCNF? … no

Third Normal Form

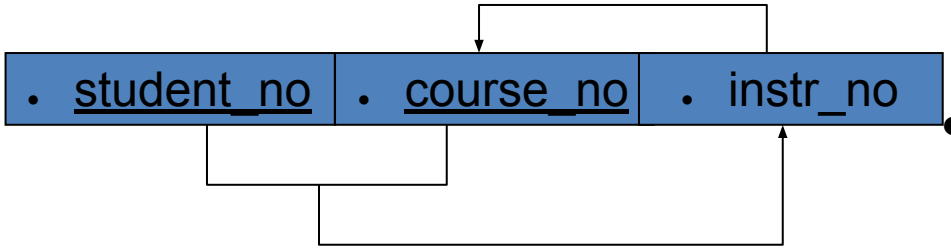


- We correct the situation by decomposing the original relation into two 3NF relations. Note the decomposition is *lossless*.



- Verify these two relations are in 3NF.

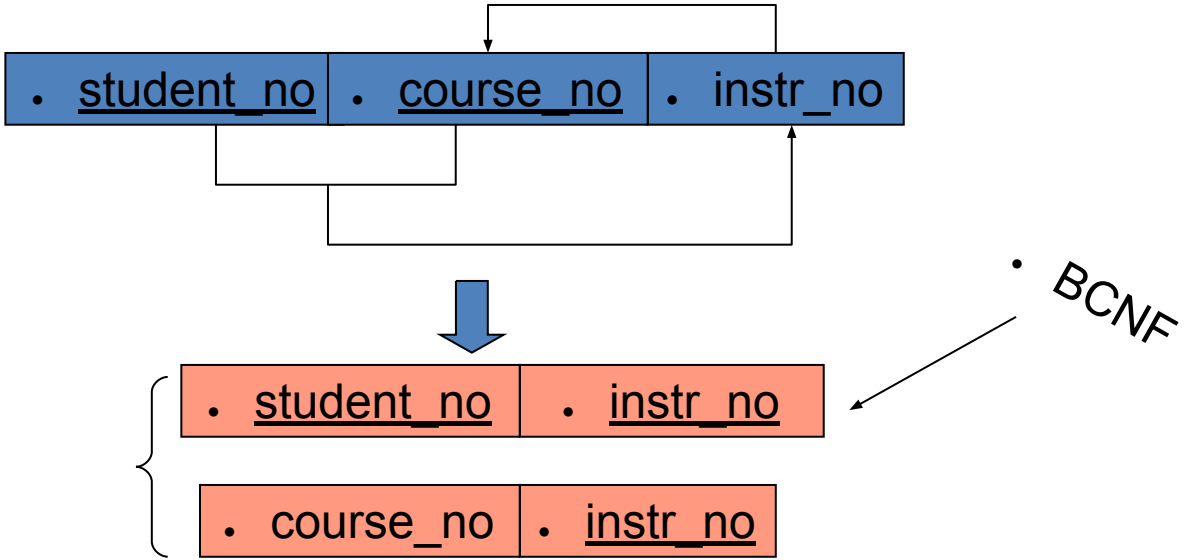
- **In 3NF, but not in BCNF:**



- *Instructor teaches one course only.*
- *Student takes a course and has one instructor.*

- {student_no, course_no} → instr_no
- instr_no → course_no

- since we have instr_no → course-no, but instr_no is not a
- Candidate key.



- {student_no, instr_no} → student_no
- {student_no, instr_no} → instr_no
- instr_no → course_no

ACID Properties

effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.

I- Isolation − In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.

D- Durability − The database should be durable enough to hold all its latest updates even if the system fails or restarts. If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.

ACID Properties

ACID Properties : To ensure the integrity of data during a transaction (A transaction is a unit of program that updates various data items), the database system must maintains the following properties.

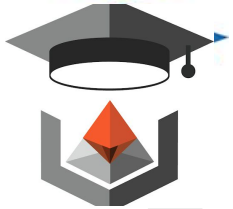
A - Atomicity – By this, we mean that either the entire transaction takes place at once or doesn't happen at all. There is no midway i.e. transactions do not occur partially. Each transaction is considered as one unit and either runs to completion or is not executed at all. It involves following two operations.
—**Abort**: If a transaction aborts, changes made to database are not visible.
—**Commit**: If a transaction commits, changes made are visible.
Atomicity is also known as the 'All or nothing rule'.

C- Consistency – The database must remain in a consistent state after any transaction. No transaction should have any adverse.

ACID Properties

Before: X : 500	Y: 200
Transaction T	
T1	**T2**
Read (X) X: = X – 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

T	T''
Read (X) X: = X*100 Write (X) Read (Y) Y: = Y – 50 Write	Read (X) Read (Y) Z: = X + Y Write (Z)



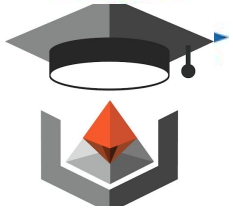
Transaction Concepts

A ***transaction*** is a *unit* of program execution that accesses and possibly updates various data items.

A transaction must see a consistent database.

During transaction execution the database may be inconsistent.

When the transaction is committed, the database must be consistent.



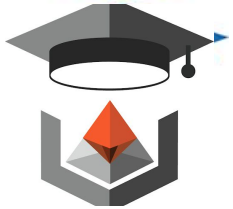
Transaction Management

If the transaction aborted, the DB must be restored to its prior state. Means such transaction must be undone or rolled back

Two main issues to deal with:

Failures of various kinds, such as hardware failures and system crashes

Concurrent execution of multiple transactions



Transaction Management

COMMIT statement – ends the SQL trans.; effects permanently recorded within DB

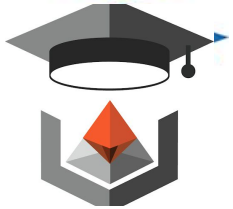
ROLLBACK statement – DB is rolled back to its previous consistent state and all the changes are aborted

Reach end of the program successfully – similar to COMMIT

Program abnormally terminated – similar to ROLLBACK

Transaction Log

- Keep track of all transactions that update the DB
- If failure occurs, information that was stored here will be used for recovery
- It is triggered by ROLL BACK statement, program abnormal termination, or system failure
- It states before-and-after data of the DB and the tables, rows and attribute values that participated in the transaction



Transaction Log

The transaction log is subject to dangers such as disk full conditions and disk crashes

It has to be managed like other DBs

Transaction log will increase the processing overhead – but it is worthwhile

Example

Transaction to transfer $50 from account *A* to account *B*:

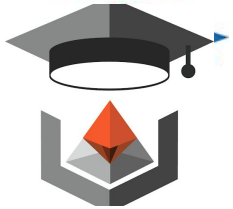
1. **read**(*A*)
2. *A* := *A* − 50
3. **write**(*A*)
4. **read**(*B*)
5. *B* := *B* + 50
6. **write**(*B*)

Consistency requirement − the sum of *A* and *B* is unchanged by the execution of the transaction.
Atomicity requirement — if the transaction fails after step 3 and before step 6, the system should ensure that its updates are not reflected in the database, else an inconsistency will result.

Example

Durability requirement — once the user has been notified that the transaction has completed (i.e., the transfer of the $50 has taken place), the updates to the database by the transaction must persist despite failures.

Isolation requirement — if between steps 3 and 6, another transaction is allowed to access the partially updated database, it will see an inconsistent database
(the sum *A + B* will be less than it should be).
Can be ensured trivially by running transactions ***serially***, that is one after the other. However, executing multiple transactions concurrently has significant benefits (this is not covered in WXES2103)



Transaction state

Active, the initial state; the transaction stays in this state while it is executing

Partially committed, after the final statement has been executed.

Failed, after the discovery that normal execution can no longer proceed.

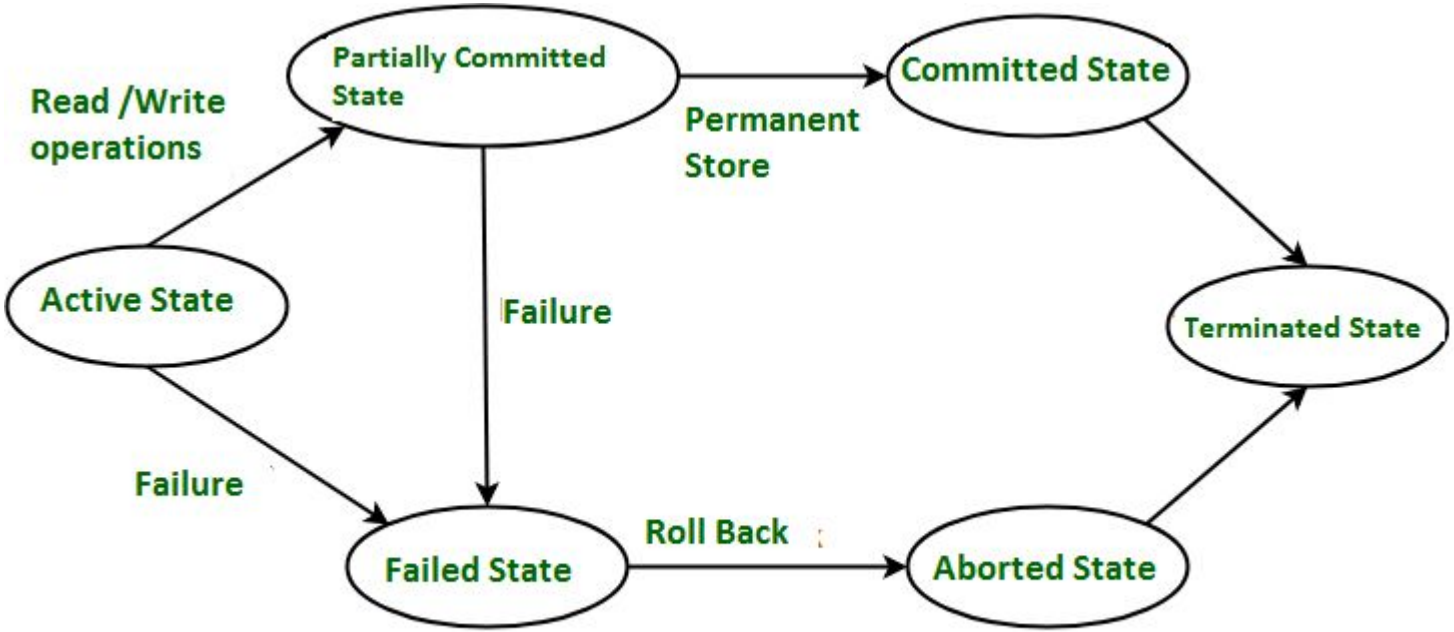
Aborted, after the transaction has been rolled back and the database restored to its state prior to the start of the transaction. Two options after it has been aborted:

restart the transaction – only if no internal logical error

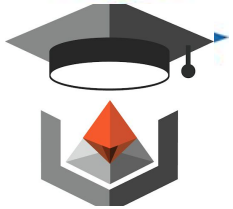
kill the transaction

Committed, after *successful completion*.

Transaction state



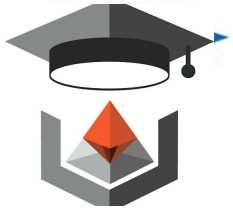
Transaction States in DBMS



Concurrency Control

Concurrency control protocols ensure atomicity, isolation, and serializability of concurrent transactions. The concurrency control protocol can be divided into three categories:

- Lock based protocol
- Time-stamp protocol
- Validation based protocol

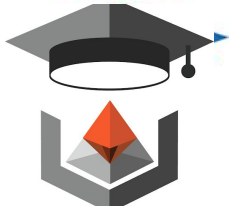


Concurrency Control

In the concurrency control, the multiple transactions can be executed simultaneously.
It may affect the transaction result. It is highly important to maintain the order of execution of those transactions.

Concurrency control protocols ensure atomicity, isolation, and serializability of concurrent transactions. The concurrency control protocol can be divided into three categories:

- Lock based protocol
- Time-stamp protocol
- Validation based protocol



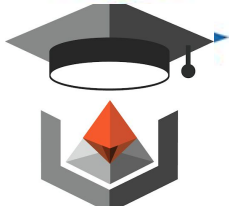
Lock Based Protocol

Shared lock :

These locks are reffered as read locks, and denoted by 'S'.
If a transaction T has obtained Shared-lock on data item X, then T can read X, but cannot write X.
Multiple Shared lock can be placed simultaneously on a data item.

Exclusive lock :

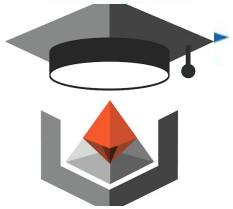
These Locks are referred as Write locks, and denoted by 'X'.
If a transaction T has obtained Exclusive lock on data item X, then T can be read as well as write X. Only one Exclusive lock can be placed on a data item at a time. This means multiple transactions does not modify the same data simultaneously.



Lock Based Protocol

1. Simplistic lock protocol
It is the simplest way of locking the data while transaction. Simplistic lock-based protocols allow all the transactions to get the lock on the data before insert or delete or update on it. It will unlock the data item after completing the transaction.

2. Pre-claiming Lock Protocol
Pre-claiming Lock Protocols evaluate the transaction to list all the data items on which they need locks.
Before initiating an execution of the transaction, it requests DBMS for all the lock on all those data items.
If all the locks are granted then this protocol allows the transaction to begin. When the transaction is completed then it releases all the lock.
If all the locks are not granted then this protocol allows the transaction to rolls back and waits until all the locks are granted.



Lock Based Protocol

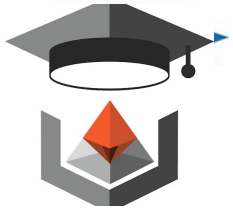
3. Two-phase locking (2PL)
The two-phase locking protocol divides the execution phase of the transaction into three parts.
In the first part, when the execution of the transaction starts, it seeks permission for the lock it requires.
In the second part, the transaction acquires all the locks. The third phase is started as soon as the transaction releases its first lock.
In the third phase, the transaction cannot demand any new locks. It only releases the acquired locks.

4. Strict Two-phase locking (Strict-2PL)
The first phase of Strict-2PL is similar to 2PL. In the first phase, after acquiring all the locks, the transaction continues to execute normally.
The only difference between 2PL and strict 2PL is that Strict-2PL does not release a lock after using it.

Lock Based Protocol

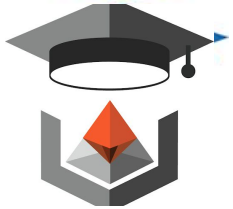
Time	Transaction	Remarks
t0	Lock - X (A)	acquire Exclusive lock on A.
t1	Read A	read original value of A
t2	A = A - 100	subtract 100 from A
t3	Write A	write new value of A
t4	Lock - X (B)	acquire Exclusive lock on B.
t5	Read B	read original value of B
t6	B = B + 100	add 100 to B
t7	Write B	write new value of B
t8	Unlock (A)	release lock on A
t9	Unock (B)	release lock on B

2PL



Timestamp Based Protocol

- The Timestamp Ordering Protocol is used to order the transactions based on their Timestamps. The order of transaction is nothing but the ascending order of the transaction creation.
- The priority of the older transaction is higher that's why it executes first. To determine the timestamp of the transaction, this protocol uses system time or logical counter.
- The lock-based protocol is used to manage the order between conflicting pairs among transactions at the execution time. But Timestamp based protocols start working as soon as a transaction is created.
- Let's assume there are two transactions T1 and T2. Suppose the transaction T1 has entered the system at 007 times and transaction T2 has entered the system at 009 times. T1 has the higher priority, so it executes first as it is entered the system first.
- The timestamp ordering protocol also maintains the timestamp of last 'read' and 'write' operation on a data.



Validation Based Protocol

Validation phase is also known as optimistic concurrency control technique. In the validation based protocol, the transaction is executed in the following three phases:

1. Read phase: In this phase, the transaction T is read and executed. It is used to read the value of various data items and stores them in temporary local variables. It can perform all the write operations on temporary variables without an update to the actual database.
2. Validation phase: In this phase, the temporary variable value will be validated against the actual data to see if it violates the serializability.
3. Write phase: If the validation of the transaction is validated, then the temporary results are written to the database or system otherwise the transaction is rolled back.

Any Question?

au.communication@accoliteindia.com

Thank You