Ternopil Ivan Pul'uj National Technical University

Department of process automation and production

LECTURE NOTES FOR DATABASE MANAGEMENT SYSTEM DBMS

(by Elena Rogatynska)

Ternopil
2014

Table of Content

THEME 1

DBMS Overview

Database is collection of data which is related by some aspect. Data is collection of facts and figures which can be processed to produce information. Name of a student, age, class and her subjects can be counted as data for recording purposes.

Mostly data represents recordable facts. Data aids in producing information which is based on facts. For example, if we have data about marks obtained by all students, we can then conclude about toppers and average marks etc.

A database management system stores data, in such a way which is easier to retrieve, manipulate and helps to produce information.

Characteristics

Traditionally data was organized in file formats. DBMS was all new concepts then and all the research was done to make it to overcome all the deficiencies in traditional style of data management.

Modern DBMS has the following characteristics:

- **Real-world entity:** Modern DBMS are more realistic and uses real world entities to design its architecture. It uses the behavior and attributes too. For example, a school database may use student as entity and their age as their attribute.
- **Relation-based tables:** DBMS allows entities and relations among them to form as tables. This eases the concept of data saving. A user can understand the architecture of database just by looking at table names etc.
- **Isolation of data and application:** A database system is entirely different than its data. Where database is said to active entity, data is said to be passive one on which the database works and organizes. DBMS also stores metadata which is data about data, to ease its own process.

- **Less redundancy:** DBMS follows rules of normalization, which splits a relation when any of its attributes is having redundancy in values. Following normalization, which itself is a mathematically rich and scientific process, make the entire database to contain as less redundancy as possible.
- **Consistency**: DBMS always enjoy the state on consistency where the previous form of data storing applications like file processing does not guarantee this. Consistency is a state where every relation in

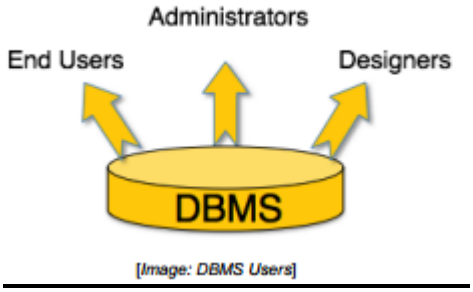
database remains consistent. There exist methods and techniques, which can detect attempt of leaving database in inconsistent state.

- **Query Language:** DBMS is equipped with query language, which makes it more effic ient to retrieve and manipulate data. A user can apply as many and different filtering options, as he or she wants. Traditionally it was not possible where file-processing system was used.
- **ACID Properties**: DBMS follows the concepts for ACID properties, which stands for Atomicity, Consistency, Isolation and Durability. These concepts are applied on transactions, which manipulate data in database. ACID properties maintains database in healthy state in multi - transactional environment and in case of failure.
- **Multiuser and Concurrent Access:** DBMS support multi-user environment and allows them to access and manipulate data in parallel. Though there are restrictions on transactions when they attempt to handle same data item, but users are always unaware of them.
- **Multiple views:** DBMS offers multiples views for different users. A user who is in sales department will have a different view of database than a person working in production department. This enables user to have a concentrate view of database according to their requirements.
- **Security**: Features like multiple views offers security at some extent where users are unable to access data of other users and departments. DBMS offers methods to impose constraints while entering data into

database and retrieving data at later stage. DBMS offers many different levels of security features, which enables multiple users to have different view with different features. For example, a user in sales department cannot see data of purchase department is one thing, additionally how much data of sales department he can see, can also be managed. Because DBMS is not saved on disk as traditional file system it is very hard for a thief to break the code.

Users

DBMS is used by various users for various purposes. Some may involve in retrieving data and some may involve in backing it up. Some of them are described as follows:



[Image: DBMS Users]

- **Administrators:** A bunch of users maintain the DBMS and are responsible for administrating the database. They are responsible to look after its usage and by whom it should be used. They create users access and apply limitation to maintain isolation and force security. Administrators also look after DBMS resources like system license, software application and tools required and other hardware related maintenance.
- **Designer**: This is the group of people who actually works on designing part of database. The actual database is started with requirement analysis followed by a good designing process. They people keep a close watch on what data should be kept and in what format. They identify and design the whole set of entities, relations, constraints and views.

- **End Users:** This group contains the persons who actually take advantage of database system. End users can be just viewers who pay attention to the logs or market rates or end users can be as sophisticated as business analysts who take the most of it.

THEME 2

DBMS Architecture

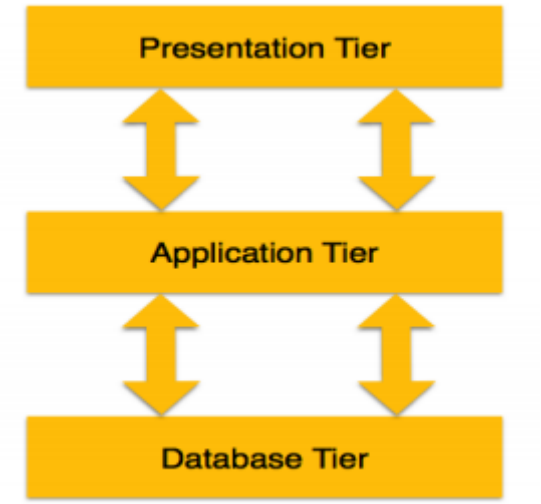
The design of a Database Management System highly depends on its architecture. It can be centralized or decentralized or hierarchical. DBMS architecture can be seen as single tier or multi tier. n-tier architecture divides the whole system into related but independent n modules, which can be independently modified, altered, changed or replaced.

In 1-tier architecture, DBMS is the only entity where user directly sits on DBMS and uses it. Any changes done here will directly be done on DBMS itself. It does not provide handy tools for end users and preferably database designer and programmers use single tier architecture.

If the architecture of DBMS is 2-tier then must have some application, which uses the DBMS. Programmers use 2-tier architecture where they access DBMS by means of application. Here application tier is entirely independent of database in term of operation, design and programming.

3-tier architecture

Most widely used architecture is 3-tier architecture. 3-tier architecture separates it tier from each other on basis of users. It is described as follows:



[Image: 3-tier DBMS architecture]

- **Database (Data) Tier:** At this tier, only database resides. Database along with its query processing languages sits in layer-3 of 3-tier architecture. It also contains all relations and their constraints.
- **Application (Middle) Tier:** At this tier the application server and program, which access database, resides. For a user this application tier works as abstracted view of database. Users are unaware of any existence of database beyond application. For database-tier, application tier is the user of it. Database tier is not aware of any other user beyond application tier. This tier works as mediator between the two.
- **User (Presentation) Tier:** An end user sits on this tier. From a user's aspect this tier is everything. He/she doesn't know about any existence or form of database beyond this layer. At this layer multiple views of database can be provided by the application. All views are generated by applications, whichreside in application tier.

Multiple tier database architecture is highly modifiable as almost all its components are independent and can be changed independently.

THEME 3

DBMS Data Models

Data model tells how the logical structure of a database is modeled. Data Models are fundamental entities to introduce abstraction in DBMS. Data models define how data is connected to each other and how it will be processed and stored inside the system.

The very first data model could be flat data-models where all the data used to be kept in same plane. Because earlier data models were not so scientific they were prone to introduce lots of duplication and update anomalies.

Entity-Relationship Model

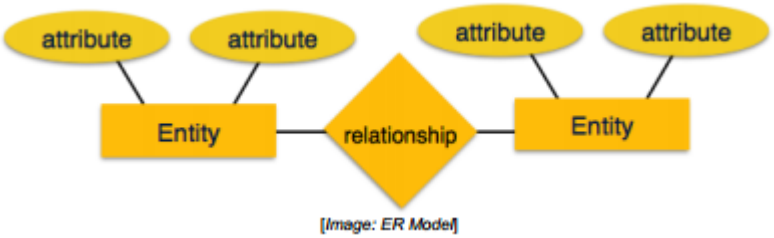
Entity-Relationship model is based on the notion of real world entities and relationship among them. While formulating real-world scenario into database model, ER Model creates entity set, relationship set, general attributes and constraints.

ER Model is best used for the conceptual design of database.

ER Model is based on:

- **Entities** and their attributes
- **Relationships** among entities

These concepts are explained below.



[Image: ER Model]

- **Entity**

An entity in ER Model is real world entity, which has some properties called attributes. Every attribute is defined by its set of values, called domain.

For example, in a school database, a student is considered as an entity. Student has various attributes like name, age and class etc.

- **Relationship**

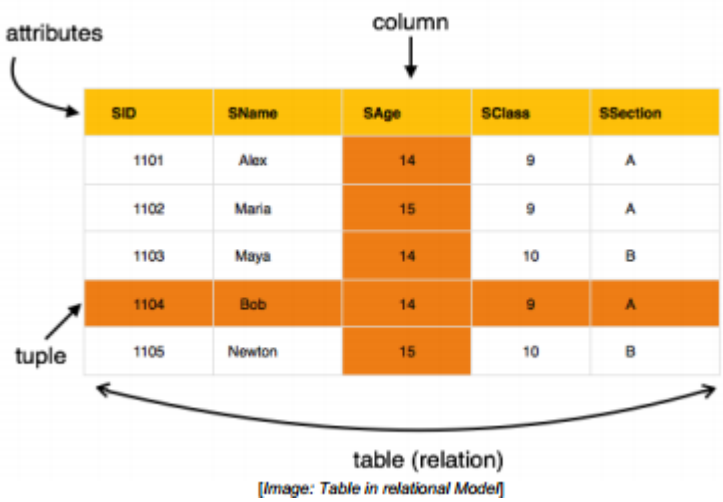
The logical association among entities is called relationship. Relationships are mapped with entities in various ways. Mapping cardinalities define the number of association between two entities.

Mapping cardinalities:

- one to one
- one to many
- many to one
- many to many

Relational Model

The most popular data model in DBMS is *Relational Model*. It is more scientific model then others. This model is based on first-order predicate logic and defines table as an n-ary relation.



[*Image: Table in relational Model*]

The main highlights of this model are:

- Data is stored in tables called relations.
- Relations can be normalized.
- In normalized relations, values saved are atomic values.
- Each row in relation contains unique value

- Each column in relation contains values from a same domain.

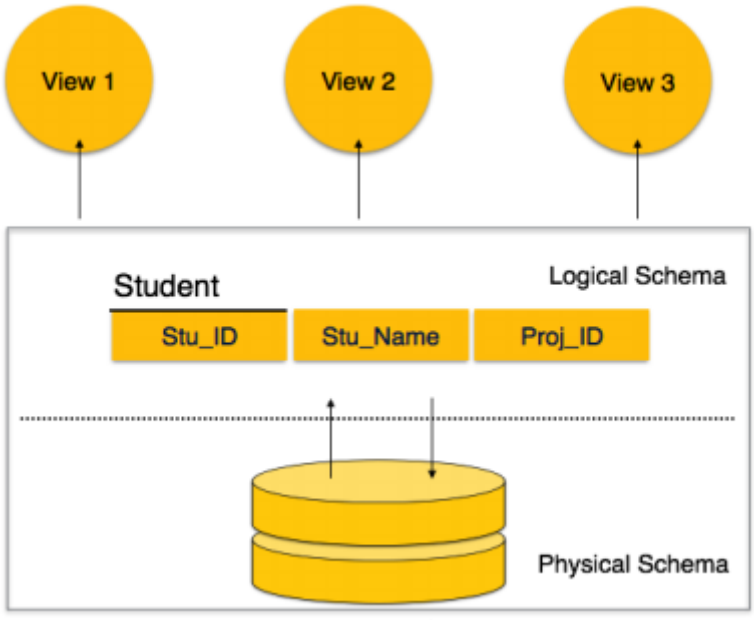
THEME 4

DBMS Data Schemas

Database schema

Database schema skeleton structure of and it represents the logical view of entire database. It tells about how the data is organized and how relation among them is associated. It formulates all database constraints that would be put on data in relations, which resides in database.

A database schema defines its entities and the relationship among them. Database schema is a descriptive detail of the database, which can be depicted by means of schema diagrams. All these activities are done by database designer to help programmers in order to give some ease of understanding all aspect of database.



[Image: Database Schemas]

Database schema can be divided broadly in two categories:

- Physical Database Schema: This schema pertains to the actual storage of data and its form of storage like files, indices etc. It defines the how data will be stored in secondary storage etc.

- Logical Database Schema: This defines all logical constraints that need to be applied on data stored. It defines tables, views and integrity constraints etc.

Database Instance

It is important that we distinguish these two terms individually. Database schema is the skeleton of database. It is designed when database doesn't exist at all and very hard to do any changes once the database is operational. Database schema does not contain any data or information.

Database instances, is a state of operational database with data at any given time. This is a snapshot of database. Database instances tend to change with time. DBMS ensures that its every instance (state) must be a valid state by keeping up to all validation, constraints and condition that database designers has imposed or it is expected from DBMS itself

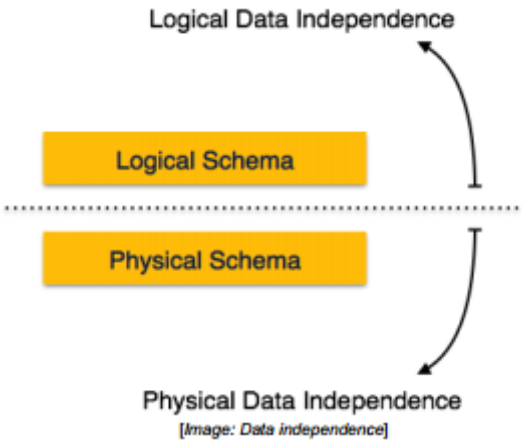
THEME 5

DBMS Data Independence

If the database system is not multi-layered then it will be very hard to make any changes in the database system. Database systems are designed in multi-layers as we leant earlier.

Data Independence:

There's a lot of data in whole database management system other than user's data. DBMS comprises of three kinds of schemas, which is in turn data about data (Meta-Data). Meta-data is also stored along with database, which once stored is then hard to modify. But as DBMS expands, it needs to be changed over the time satisfy the requirements of users. But if the whole data were highly dependent it would become tedious and highly complex



[Image: Data independence]

Data about data itself is divided in layered architecture so that when we change data at one layer it does not

affect the data layered at different level. This data is independent but mapped on each other.

Logical Data Independence

Logical data is data about database, that is, it stores information about how data is managed inside. For example, a table (relation) stored in the database and all constraints, which are applied on that relation.

Logical data independence is a kind of mechanism, which liberalizes itself from actual data stored on the disk. If we do some changes on table format it should not change the data residing on disk.

Physical Data Independence

All schemas are logical and actual data is stored in bit format on the disk. Physical data independence is the power to change the physical data without impacting the schema or logical data.

For example, in case we want to change or upgrade the storage system itself, that is, using SSD instead of Hard-disks should not have any impact on logical data or schemas.

THEME 6

ER Model : Basic Concepts

Entity relationship model defines the conceptual view of database. It works around real world entity and association among them. At view level, ER model is considered well for designing databases.

Entity

A real-world thing either animate or inanimate that can be easily identifiable and distinguishable. For example, in a school database, student, teachers, class and course offered can be considered as entities. All entities have some attributes or properties that give them their identity.

An entity set is a collection of similar types of entities. Entity set may contain entities with attribute sharing similar values. For example, Students set may contain all the student of a school; likewise Teachers set may contain all the teachers of school from all faculties. Entities sets need not to be disjoint.

Attributes

Entities are represented by means of their properties, called attributes. All attributes have values. *For example*, a student entity may have name, class, age as attributes.

There exist a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.

Types of attributes:

- **Simple attribute:**

Simple attributes are atomic values, which cannot be divided further. For example, student's phone-number is an atomic value of 10 digits.

- **Composite attribute:**

Composite attributes are made of more than one simple attribute. For example, a student's complete name may have first_name and last_name.

- **Derived attribute:**

Derived attributes are attributes, which do not exist physical in the database, but there values are derived from other attributes presented in the database. For example, average_salary in a department should be saved in database instead it can be derived. For another example, age can be derived from data_of_birth.

- **Single-valued attribute:**

Single valued attributes contain on single value. For example: Social_Security_Number.

- **Multi-value attribute:**

Multi-value attribute may contain more than one values. For example, a person can have more than one phone numbers, email_addresses etc.

These attribute types can come together in a way like:

- simple single-valued attributes
- simple multi-valued attributes
- composite single-valued attributes
- composite multi-valued attributes

Entity-set and Keys

Key is an attribute or collection of attributes that uniquely identifies an entity among entity set.

For example, roll_number of a student makes her/him identifiable among students.

- **Super Key:** Set of attributes (one or more) that collectively identifies an entity in an entity set.
- **Candidate Key:** Minimal super key is called candidate key that is, supers keys for which no proper subset are a superkey. An entity set may have more than one candidate key.

- **Primary Key:** This is one of the candidate key chosen by the database designer to uniquely identify the entity set.

Relationship

The association among entities is called relationship. For example, employee entity has relation works_at with department. Another example is for student who enrolls in some course. Here, Works_at and Enrolls are called *relationship*.

Relationship Set:

Relationship of similar type is called relationship set. Like entities, a relationship too can have attributes. These attributes are called descriptive attributes.

Degree of relationship

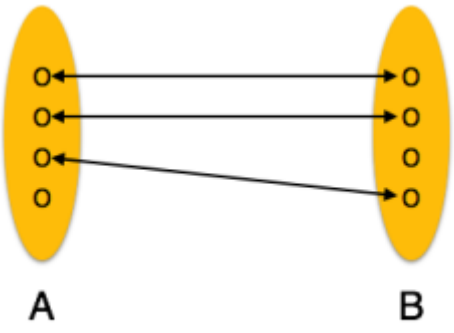
The number of participating entities in an relationship defines the degree of the relationship.

- Binary = degree 2
- Ternary = degree 3
- *n*-ary = degree

Mapping Cardinalities:

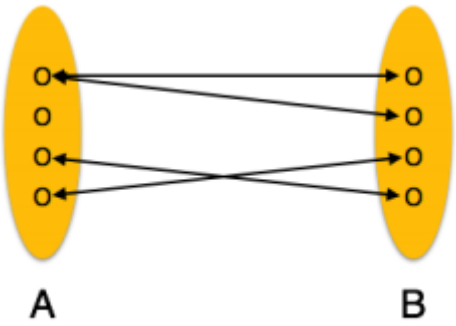
Cardinality defines the number of entities in one entity set which can be associated to the number of entities of other set via relationship set.

- **One-to-one:** one entity from entity *set A* can be associated with at most one entity of entity *set B* and vice versa.



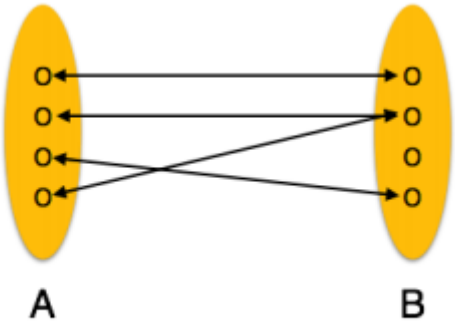
[Image: One-to-one relation]

- **One-to-many:** One entity from entity *set A* can be associated with more than one entities of entity *set B* but from entity set B one entity can be associated with at most one entity.



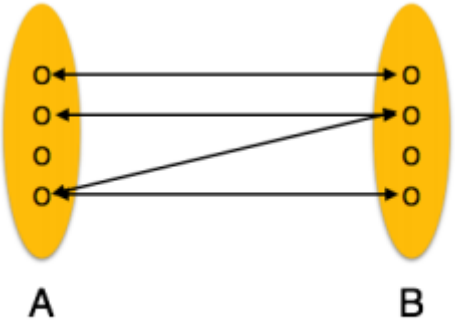
[*Image: One-to-many relation*]

- **Many-to-one:** More than one entities from entity *set A* can be associated with at most one entity of entity *set B* but one entity from entity *set B* can be associated with more than one entity from entity *set A*.



[*Image: Many-to-one relation*]

- **Many-to-many:** one entity from *A* can be associated with more than one entity from *B* and vice versa.



[*Image: Many-to-many relation*]

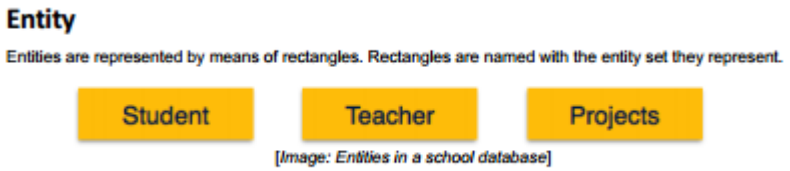
THEME 7

ER Diagram Representation

Now we shall learn how ER Model is represented by means of ER diagram. Every object like entity, attributes of an entity, relationship set, and attributes of relationship set can be represented by tools of ER diagram.

Entity

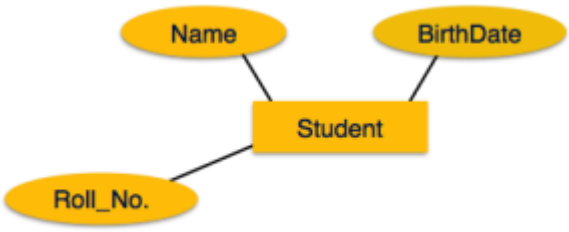
Entities are represented by means of rectangles. Rectangles are named with the entity set they represent.



[Image: Entities in a school database]

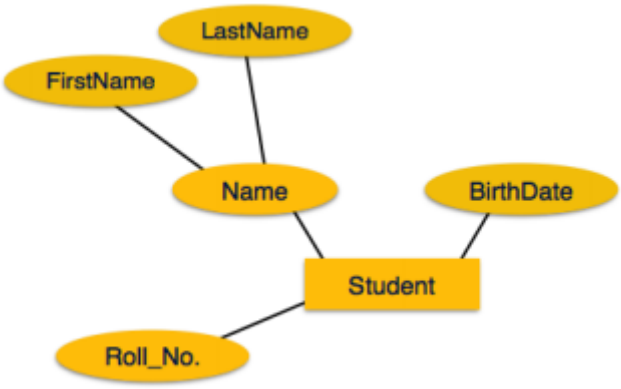
Attributes

Attributes are properties of entities. *Attributes* are represented by means of eclipses. Every eclipse represents one attribute and is directly connected to its entity (rectangle).



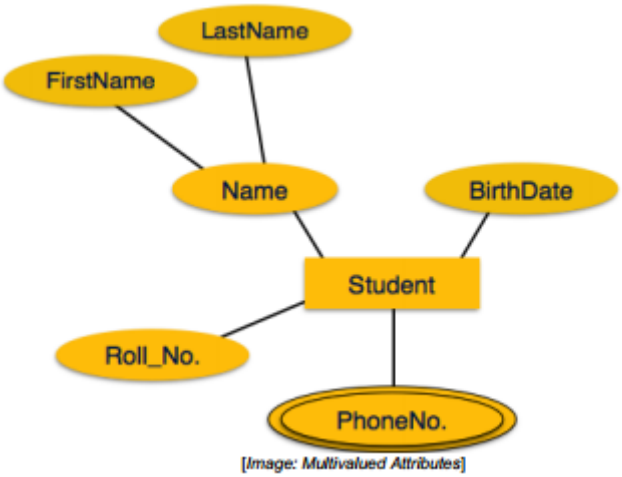
[Image: Simple Attributes]

If the attributes are composite, they are further divided in a tree like structure. Every node is then connected to its attribute. That is composite attributes are represented by eclipses that are connected with an eclipse.



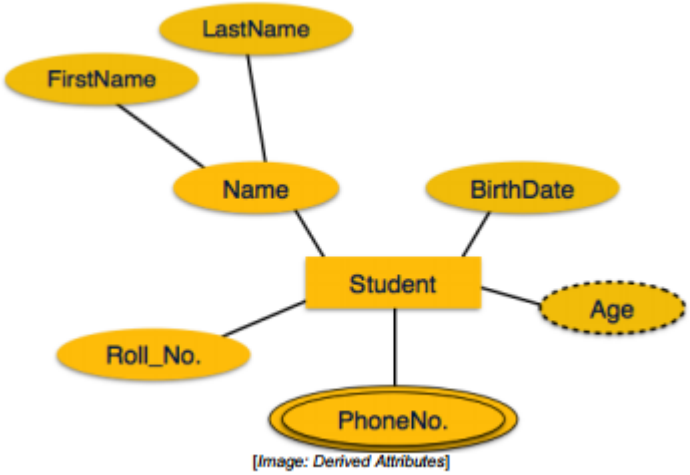
[Image: Composite Attributes]

Multivalued attributes are depicted by double eclipse.



[Image: Multivalued Attributes]

Derived attributes are depicted by dashed eclipse.



[Image: Derived Attributes]

Relationship

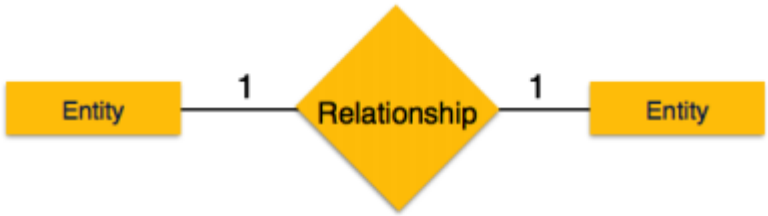
Relationships are represented by diamond shaped box. Name of the relationship is written in the diamond-box. All entities (rectangles), participating in relationship, are connected to it by a line.

Binary relationship and cardinality

A relationship where two entities are participating, is called a binary relationship. Cardinality is the number of instance of an entity from a relation that can be associated with the relation.

- **One-to-one**

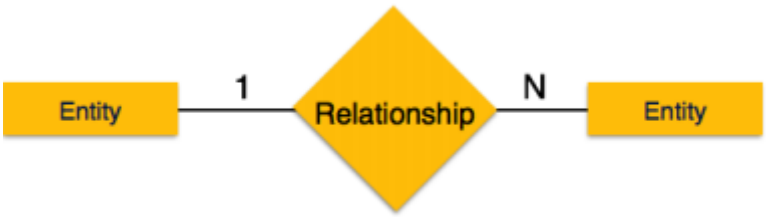
When only one instance of entity is associated with the relationship, it is marked as '*1*'. This image below reflects that only 1 instance of each entity should be associated with the relationship. It depicts one-to-one relationship.



[Image: One-to-one]

- **One-to-many**

When more than one instance of entity is associated with the relationship, it is marked as '*N*'. This image below reflects that only 1 instance of entity on the left and more than one instance of entity on the right can be associated with the relationship. It depicts one-to-many relationship.

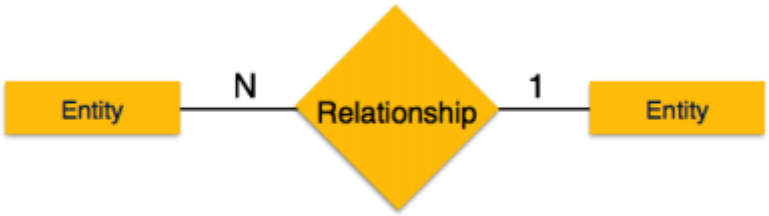


[Image: One-to-many]

- **Many-to-one**

When more than one instance of entity is associated with the relationship, it is marked as '*N*'. This image below reflects that more than one instance of entity on

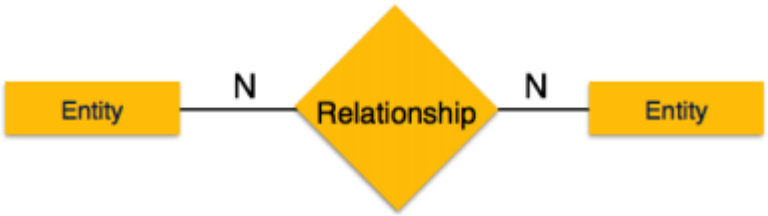
the left and only one instance of entity on the right can be associated with the relationship. It depicts many-to-one relationship.



[Image: Many-to-one]

- **Many-to-many**

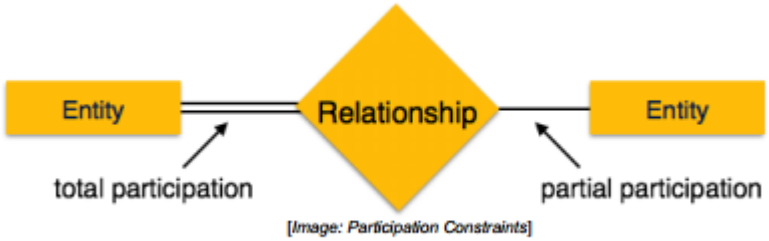
This image below reflects that more than one instance of entity on the left and more than one instance of entity on the right can be associated with the relationship. It depicts many-to-many relationship.



[Image: Many-to-many]

Participation Constraints

- **Total Participation:** Each entity in the entity is involved in the relationship. Total participation is represented by double lines.
- **Partial participation:** Not all entities are involved in the relationship. Partial participation is represented by single line.



[Image: Participation Constraints]

THEME 8

Codd's 12 Rules

Dr Edgar F. Codd did some extensive research in Relational Model of database systems and came up with twelve rules of his own which according to him, a database must obey in order to be a true relational database.

These rules can be applied on a database system that is capable of managing is stored data using only its relational capabilities. This is a foundation rule, which provides a base to imply other rules on it.

Rule 1: Information rule

This rule states that all information (data), which is stored in the database, must be a value of some table cell. Everything in a database must be stored in table formats. This information can be user data or meta-data.

Rule 2: Guaranteed Access rule

This rule states that every single data element (value) is guaranteed to be accessible logically with combination of table-name, primary-key (row value) and attribute-name (column value). No other means, such as pointers, can be used to access data.

Rule 3: Systematic Treatment of NULL values

This rule states the NULL values in the database must be given a systematic treatment. As a NULL may have several meanings, i.e. NULL can be interpreted as one the following: data is missing, data is not known, data is not applicable etc.

Rule 4: Active online catalog

This rule states that the structure description of whole database must be stored in an online catalog, i.e. data dictionary, which can be accessed by the authorized users. Users can use the same query language to access the catalog which they use to access the database itself.

Rule 5: Comprehensive data sub-language rule

This rule states that a database must have a support for a language which has linear syntax which is capable of data definition, data manipulation and transaction management operations. Database can be accessed by means of this language only, either directly or by means of some application. If the database can be accessed or manipulated in some way without any help of this language, it is then a violation.

Rule 6: View updating rule

This rule states that all views of database, which can theoretically be updated, must also be updatable by the system.

Rule 7: High-level insert, update and delete rule

This rule states the database must employ support high-level insertion, updation and deletion. This must not be limited to a single row that is, it must also support union, intersection and minus operations to yield sets of
data records.

Rule 8: Physical data independence

This rule states that the application should not have any concern about how the data is physically stored. Also, any change in its physical structure must not have any impact on application.

Rule 9: Logical data independence

This rule states that the logical data must be independent of its user's view (application). Any change in logical data must not imply any change in the application using it. For example, if two tables are merged or one is split into two different tables, there should be no impact the change on user application. This is one of the most difficult rule to apply.

Rule 10: Integrity independence

This rule states that the database must be independent of the application using it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes database independent of the front-end application and its interface.

Rule 11: Distribution independence

This rule states that the end user must not be able to see that the data is distributed over various locations. User must also see that data is located at one site only. This rule has been proven as a foundation of distributed database systems.

Rule 12: Non-subversion rule

This rule states that if a system has an interface that provides access to low level records, this interface then must not be able to subvert the system and bypass security and integrity constraints.

THEME 9

Relation Data Model

Relational data model is the primary data model, which is used widely around the world for data storage and processing. This model is simple and have all the properties and capabilities required to process data with storage efficiency.

Concepts

Tables: In relation data model, relations are saved in the format of Tables. This format stores the relation among entities. A table has rows and columns, where rows represent records and columns represents the attributes.

Tuple: A single row of a table, which contains a single record for that relation is called a tuple.

Relation instance: A finite set of tuples in the relational database system represents relation instance. *Relation instances* do not have duplicate tuples.

Relation schema: This describes the relation name (table name), attributes and their names.

Relation key: Each row has one or more attributes which can identify the row in the relation (table) uniquely, is called the relation key.

Attribute domain: Every attribute has some pre-defined value scope, known as attribute domain.

Constraints

Every relation has some conditions that must hold for it to be a valid relation. These conditions are called Relational Integrity Constraints. There are three main integrity constraints.

- Key Constraints
- Domain constraints
- Referential integrity constraints

Key Constraints:

There must be at least one minimal subset of attributes in the relation, which can identify a tuple uniquely. This minimal subset of attributes is called key for that relation. If there are more than one such minimal subsets, these are called candidate keys.

Key constraints forces that:

- in a relation with a key attribute, no two tuples can have identical value for key attributes.
- key attribute can not have NULL values.

Key constrains are also referred to as Entity Constraints.

Domain constraints

Attributes have specific values in real-world scenario. For example, age can only be positive integer. The same constraints has been tried to employ on the attributes of a relation. Every attribute is bound to have a specific range of values. For example, age can not be less than zero and telephone number can not be a outside *0 -9.*

Referential integrity constraints

This integrity constraints works on the concept of Foreign Key. A key attribute of a relation can be referred in other relation, where it is called foreign key.

Referential integrity constraint states that if a relation refers to an key attribute of a different or same relation, that key element must exists.

THEME 10

Relational Algebra

Relational database systems are expected to be equipped by a query language that can assist its user to query the database instances. This way its user empowers itself and can populate the results as required. There are two kinds of query languages, relational algebra and relational calculus.

Relational algebra

Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses opera tors to perform queries. An operator can be either unary or binary. They accept relations as their input and yields relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

Fundamental operations of Relational algebra:

- Select
- Project
- Union
- Set different
- Cartesian product
- Rename

These are defined briefly as follows:

Select Operation (σ)

Selects tuples that satisfy the given predicate from a relation.

Notation $\sigma_p(r)$

Where *p* stands for selection predicate and *r* stands for relation. *p* is prepositional logic formulae which may use connectors like and, or and not. These terms may use relational operators like: =, ≠, ≥, < , >, ≤.

For example:

$$\sigma_{subject="database"}(Books)$$

Output: Selects tuples from books where subject is '*database*'.

$$\sigma_{subject="database" \text{ and } price="450"}(Books)$$

Output: Selects tuples from books where subject is 'database' and 'price' is 450.

$$\sigma_{subject="database" \text{ and } price < "450" \text{ or } year > "2010"}(Books)$$

Output: Selects tuples from books where subject is 'database' and 'price' is 450 or the publication year is greater than 2010, that is published after 2010.

Project Operation (∏)

Projects column(s) that satisfy given predicate.

Notation: $\prod_{A1, A2, An} (r)$

Where a_1, a_2 , a_n are attribute names of relation r.

Duplicate rows are automatically eliminated, as relation is a set.

for example:

$$\prod_{subject, author} (Books)$$

Selects and projects columns named as subject and author from relation Books.

Union Operation (∪)

Union operation performs binary union between two given relations and is defined as:

$$r \cup s = \{ t \mid t \in r \text{ or } t \in s \}$$

Notion: *r U s*

Where *r* and *s* are either database relations or relation result set (temporary relation).

For a union operation to be valid, the following conditions must hold:

- *r*, *s* must have same number of attributes.
- Attribute domains must be compatible.

Duplicate tuples are automatically eliminated.

$$\prod_{\text{author}} (\text{Books}) \cup \prod_{\text{author}} (\text{Articles})$$

Output: Projects the name of author who has either written a book or an article or both.

Set Difference (−)

The result of set difference operation is tuples which present in one relation but are not in the second relation.

Notation: *r* − *s*

Finds all tuples that are present in *r* but not *s*.

$$\prod_{\text{author}} (\text{Books}) - \prod_{\text{author}} (\text{Articles})$$

Output: Results the name of authors who has written books but not articles.

Cartesian Product (X)

Combines information of two different relations into one.

Notation: *r* X *s*

Where *r* and *s* are relations and there output will be defined as:

r X *s* = { *q t* | *q* ∈ *r* and *t* ∈ *s*}

$$\prod_{\text{author} = \text{'tutorialspoint'}} (\text{Books X Articles})$$

Output : yields a relation as result which shows all books and articles written by tutorialspoint.

Rename operation (ρ)

Results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. rename operation is denoted with small greek letter rho ρ

Notation: ρ x (E)

Where the result of expression *E* is saved with name of *x*.

Additional operations are:

- Set intersection
- Assignment
- Natural join

Relational Calculus

In contrast with Relational Algebra, Relational Calculus is non -procedural query language, that is, it tells what to do but never explains the way, how to do it.

Relational calculus exists in two forms:

Tuple relational calculus (TRC)

Filtering variable ranges over tuples

Notation: { T | Condition }

Returns all tuples T that satisfies condition.

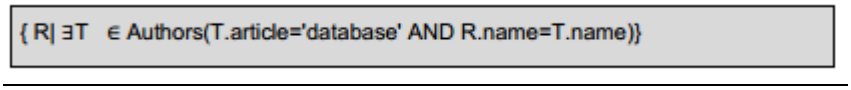
For Example:

```
{ T.name |  Author(T) AND T.article = 'database' }
```

Output: returns tuples with 'name' from *Author* who has written article on 'database'.

TRC can be quantified also. We can use Existential (∃) and Universal Quantifiers (∀).

For example:



Output : the query will yield the same result as the previous one.

Domain relational calculus (DRC)

In DRC the filtering variable uses domain of attributes instead of entire tuple values (as done in TRC, mentioned above).

Notation:

$\{ a_1, a_2, a_3, ..., a_n \mid P (a_1, a_2, a_3, ... ,a_n)\}$

where a_1, a_2 are attributes and *P* stands for formulae built by inner attributes.

For example:

Output: *Yields Article*, *Page* and *Subject* from relation where *Subject* is database.

Just like TRC, DRC also can be written using existential and universal quantifiers. DRC also involves relational operators.

Expression power of *Tuple* relation calculus and *Domain* relation calculus is equivalent to Relational Algebra.

THEME 11

DBMS Normalization

Functional Dependency

Functional dependency (*FD*) is set of constraints between two attributes in a relation. Functional dependency says that if two tuples have same values for attributes **A1, A2,..., An** then those two tuples must have to have same values for attributes **B1, B2, ..., Bn.**

Functional dependency is represented by arrow sign (→), that is **X→Y**, where **X** functionally determines **Y**. The left hand side attributes determines the values of attributes at right hand side.

Armstrong's Axioms

If **F** is set of functional dependencies then the closure of **F**, denoted as $\mathbf{F}^+$, is the set of all functional dependencies logically implied by **F**. Armstrong's Axioms are set of rules, when applied repeatedly generates closure of functional dependencies.

- Reflexive rule: If alpha is a set of attributes and beta is_subset_of alpha, then alpha holds beta.
- Augmentation rule: if **a → b** holds and y is attribute set, then **ay → by** also holds. That is adding attributes in dependencies, does not change the basic dependencies.
- Transitivity rule: Same as transitive rule in algebra, if **a → b** holds and **b → c** holds then **a → c** also hold. a → b is called as a *functionally determines b.*

Trivial Functional Dependency

- Trivial: If an FD **X → Y** holds where **Y** subset of **X,** then it is called a trivial FD. Trivial FDs are always hold.
- Non-trivial: If an FD **X → Y** holds where **Y** is not subset of **X**, then it is called non-trivial FD.
- Completely non-trivial: If an FD **X → Y** holds where x intersect **Y = Φ,** is said to be completely non -trivial FD.

Normalization

If a database design is not perfect it may contain anomalies, which are like a bad dream for database itself. Managing a database with anomalies is next to impossible.

- **Update anomalies:** if data items are scattered and are not linked to each other properly, then there may be instances when we try to update one data item that has copies of it scattered at several places, few instances of it get updated properly while few are left with there old values. This leaves database in an inconsistent state.
- **Deletion anomalies:** we tried to delete a record, but parts of it left undeleted because of unawareness, the data is also saved somewhere else.
- **Insert anomalies:** we tried to insert data in a record that does not exist at all.

Normalization is a method to remove all these anomalies and bring database to consistent state and free from any kinds of anomalies.

First Normal Form:

This is defined in the definition of relations (tables) itself. This rule defines that all the attributes in a relation must have atomic domains. Values in atomic domain are indivisible units.

Course	Content
Programming	Java, c++
Web	HTML, PHP, ASP

[Image: Unorganized relation]

We re-arrange the relation (table) as below, to convert it to *First Normal Form.*

Course	Content
Programming	Java
Programming	c++
Web	HTML
Web	PHP
Web	ASP

[Image: Relation in 1NF]

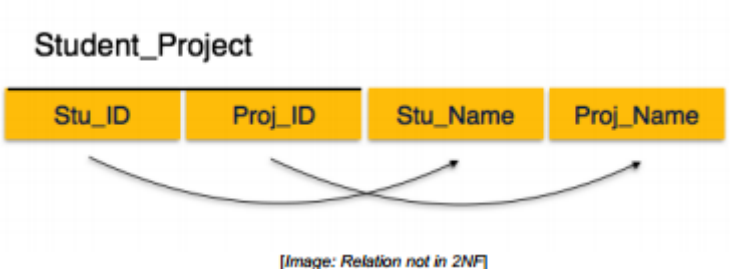
Each attribute must contain only single value from its pre-defined domain.

Second Normal Form:

Before we learn about *second normal form*, we need to understand the following:

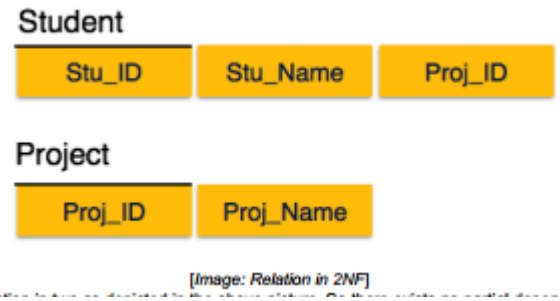
- **Prime attribute:** an attribute, which is part of prime-key, is prime attribute.
- **Non-prime attribute:** an attribute, which is not a part of prime-key, is said to be a non-prime attribute.

Second normal form says, that every non-prime attribute should be fully functionally dependent on prime key attribute. That is, if **X** → **A** holds, then there should not be any proper subset **Y** of **X**, for that **Y** → **A** also holds.



[Image: Relation not in 2NF]

We see here in **Student_Project** relation that the prime key attributes are **Stu_ID** and **Proj_ID**. According to the rule, non-key attributes, i.e. **Stu_Name** and **Proj_Name** must be dependent upon both and not on any of the prime key attribute individually. But we find that **Stu_Name** can be identified by **Stu_ID** and **Proj_Name** can be identified by **Proj_ID** independently. This is called *partial dependency*, which is not allowed in *Second Normal Form*.



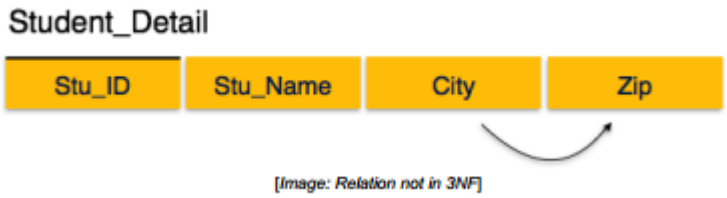
[Image: Relation in 2NF]

We broke the relation in two as depicted in the above picture. So there exists no partial dependency.

Third Normal Form:

For a relation to be in *Third Normal Form*, it must be in *Second Normal form* and the following must satisfy:

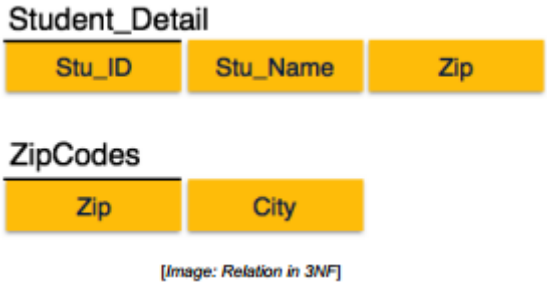
- No non-prime attribute is transitively dependent on prime key attribute
- For any non-trivial functional dependency, **X** → **A**, then either
- **X** is a superkey or,
- **A** is prime attribute.



[Image: Relation not in 3NF]

We find that in above depicted **Student_detail** relation, **Stu_ID** is *key* and only *prime key attribute*. We find that **City** can be identified by **Stu_ID** as well as **Zip** itself. Neither **Zip** is a *superkey* nor **City** is a prime attribute.

Additionally, **Stu_ID** → **Zip** → **City**, so there exists transitive dependency.



[Image: Relation in 3NF]

We broke the relation as above depicted two relations to bring it into **3NF**.

Boyce-Codd Normal Form:

BCNF is an extension of *Third Normal Form* in strict way. *BCNF* states that

- For any non-trivial functional dependency, **X** → **A**, then **X** must be a super-key.

In the above depicted picture, **Stu_ID** is super-key in **Student_Detail** relation and **Zip** is super-key in **ZipCodes** relation. So,

Stu_ID → **Stu_Name, Zip**

And

Zip → City

Confirms, that both relations are in **BCNF**.

THEME 12

SQL Overview

SQL is a programming language for **Relational Databases.** It is designed over relational algebra and tuple relational calculus. SQL comes as a package with all major distributions of RDBMS.

SQL comprises both data definition and data manipulation languages. Using the data definition properties of SQL, one can design and modify database schema whereas data manipulation properties allows SQL to store and retrieve data from database.

Data definition Language

SQL uses the following set of commands to define database schema:

CREATE

Creates new databases, tables and views from RDBMS

For example:

```
Create database tutorialspoint;
Create table article;
Create view for_students;
```

DROP

Drop commands deletes views, tables and databases from RDBMS

```
Drop object_type object_name;
Drop database tutorialspoint;
Drop table article;
Drop view for_students;
```

ALTER

Modifies database schema

```
Alter object_type object_name parameters;
```

for example

```
Alter table article add subject varchar;
```

This command adds an attribute in relation article with name subject of string type.

Data Manipulation Language

SQL is equipped with **data manipulation language**. DML modifies the database instance by inserting, updating and deleting its data. DML is responsible for all data modification in databases. SQL contains the following set of command in DML section:

- SELECT/FROM/WHERE
- INSERT INTO/VALUES
- UPDATE/SET/WHERE
- DELETE FROM/WHERE

These basic constructs allows database programmers and users to enter data and information into the database and retrieve efficiently using a number of filter options.

SELECT/FROM/WHERE

- **SELECT**

This is one of the fundamental query command of SQL. It is similar to projection operation of relational algebra. It selects the attributes based on the condition described by WHERE clause.

- **FROM**

This clause takes a relation name as an argument from which attributes are to be selected/projected. In case more than one relation names are given this clause corresponds to cartesian product.

- **WHERE**

This clause defines predicate or conditions which must match in order to qualify the attributes to be projected.

For example:

```
Select author_name
From book_author
Where age > 50;
```

This command will project names of author's from book_author relation whose age is greater than 50.

INSERT INTO/VALUES

This command is used for inserting values into rows of table (relation).

Syntax is

```
INSERT INTO table (column1 [, column2, column3 ... ]) VALUES (value1 [, value2, value3 ... ])
```

UPDATE/SET/WHERE

This command is used for updating or modifying values of columns of table (relation).

Syntax is

```
UPDATE table_name SET column_name = value [, column_name = value ...]
[WHERE condition]
```

DELETE/FROM/WHERE

This command is used for removing one or more rows from table (relation).

Syntax is

```
DELETE FROM tutorialspoints

WHERE Author="unknown";
```