SUBJECT : DATABASE SYSTEMS

SUBJECT CODE : 18K5CSELCS1:A

UNIT : III, IV,V

STAFF NAME : 1. S.RAMYA

2.J.SHANMUGHAPRIYA

Semester – V
MBE –I

Hours - 6
Credit – 5

Database Systems (18K5CSELCS1:A)

Objective: To impart knowledge in Database concepts with an overview on SQL and Oracle.

UNIT I : Introduction : Database System Applications – Purpose of Database Systems – View of Data – Database Languages – Relational Databases – Database Design – Data Storage And Querying - Transaction Management – Database Architecture – Data Mining and Infromation Retrieval – Specialty Databases - Database User and Administrators History of Database Systems.

UNIT II : Relational Databases : Introduction to the Relational Model : Structure of Relational Databases – Database Schema - Keys – Schema Diagrams – Relational Query Languages – Relational Operations.

UNIT III : Introduction to SQL : Overview of the SQL Query Language – SQL Data Definition Basic Structure of SQL Queries – Additional Basic Operations - Set Operations – Null Values – Aggregate Functions - Nested Subqueries - Modification of the Database. Intermediate SQL : Join Expressions – Views - Transactions – Integrity Constraints - SQL Data Types and Schemas – Authorization.

UNIT IV : Database Design and E-R Model – Overview of the Design Process – The Entity-Relationship Model Constraints – Removing Redundant Attributes in Entity Sets - Entity-Relationship Diagrams. Relational Database Design: Features of Good Relational Design - Atomic Domains And First Normal Form – Decomposition using Functional dependencies.

UNIT V : Critical Database Concept : Oracle Sharing Knowledge and Success: The Cooperative approach – Familiar Language of Oracle. The dangers in a Relational DataBase – Codes, Abbreviation and Naming Standards – Normalizing Names – Understanding the Data. Basic Parts of Speech in SQL : Style – Logic and Value – Combining Logic.

TEXT: Unit I - IV

"Database System Concepts" – Abraham Silberschatz, Henry F.Korth, S.Sudarshan– McGraw Hill Education (India) Pvt. Ltd., – Sixth Edition - Second Reprint -2013.

Chapters : 1, 2, 3, 4, 7.1 - 7.5, 8.1 - 8.3.

Unit – V

"Oracle 9i – The Complete Reference" – Kevin Loney, George Koch And the Experts at TUSC – Tata McGraw-Hill Edition – 2002.

Chapters: 1 – 3 (Relevent Topics Only).

Reference:

1. "Database Management System" – Ramakrishnam – McGraw – Hill Higher Education - Third Edition - 2002.
2. "Database Management System the Complete Book"- Jennifer widom,Jeffrey D. Ullman,Hector Garcia –molina Pearson - First Edition - 2003.
3. "Database Management System" - G.K.Gupta – Tata Mcgraw Hill Education Private Limited - First Edition - 2011.

UNIT III

Data Definition Language in SQL

SQL - **Structured Query Language**. It is termed as a 'query language', it defines the schema of the database, it modifies the database and also it can be used to specify special security constraints on the data in the database.

SQLs data definition language (DDL) defines not only a set of relations but also information about each relation in a database.

They are as follows:

- The schema for each relation
- The types of values associated with each attribute.
- The integrity constraints.
- The set of indices to be maintained for each relation.
- The security and authorization information for each relation.
- The physical storage structure of each relation on disk.

SQL Data Types

The SQL standard supports a variety of built-in types, including:

• char(n): A fixed-length character string with user-specified length n. The full form, character, can be used instead.

• varchar(n): A variable-length character string with user-specified maximum length n. The full form, character varying, is equivalent.

• int: An integer (a finite subset of the integers that is machine dependent). The full form, integer, is equivalent. • smallint: A small integer (a machine-dependent subset of the integer type).

• numeric(p, d): A fixed-point number with user-specified precision. The number consists of p digits (plus a sign), and d of the p digits are to the right of the decimal

point. Thus, numeric(3,1) allows 44.5 to be stored exactly, but neither 444.5 or 0.32 can be stored exactly in a field of this type.

• real, double precision: Floating-point and double-precision floating-point numbers with machine-dependent precision.

• float(n): A floating-point number, with precision of at least n digits.

SQL Schema Definition:

SQL DDL defines the schema of a relation using a command **create table**. The general form of 'create table' command is as follow:

***Create table** r(A1 D1, 2 D2, ……An Dn, (integrity constraint1), (integrity constraint$_n$));*

r is the name of the relation; $\mathbf{A_i}$ is the name of an attribute in relation r; **Di** is the domain of the attribute. The create command is completed using a semicolon.

Example: ***create table department (dept name varchar (20), building varchar (15), budget numeric (12,2), primary key (dept name));***

SQL data definition **integrity constraints** are logical condition applied to the relation for maintaining the consistency of the data in the database.

To load data into a relation **insert** command is used.

Example*:* ***insert into instructor values (10211, 'Smith', 'Biology', 66000);***

To delete any tuple or entity from a relation **delete** command is used, the general form of delete command is:

***delete from** r **where** P;*

Example: ***delete from student;***

To delete an entire relation, from the database then we can use **drop** table command.

***drop table** r;*

The alter table command adds an attribute to an already existing relation in a database. Its general form is

Alter table r add A D;
//*where A is an attribute name and D is its corresponding domain.*

Similarly, we can delete an attribute from an already existing relation in a database using **alter drop** command. The general form of alter drop command is:

Alter table r drop A;
// A is an attribute to be dropped from relation A

Basic Structure of SQL queries

The basic structure of an SQL query consists of three clauses: select, from, and where.

• The select clause is used to list the attributes desired in the result of a query.

• The from clause is a list of the relations to be accessed in the evaluation of the query.

• The where clause is a predicate involving attributes of the relation in the from clause.

A typical SQL query has the form select A1, A2, . . . , An from r1, r2, . . . ,rm where P; Each Ai represents an attribute, and each ri a relation. P is a predicate. If the where clause is omitted, the predicate P is true.

Queries on single relation:

To "Find the department names of all instructors," which can be written as:

select dept _name from instructor;

To eliminate the duplication of department name:

select distinct dept _name from instructor;

SQL allows the use of the logical connectives and, or, and not in the where clause. The operands of the logical connectives can be expressions involving the comparison operators <=, >, >=, =, and <>

Queries on Multiple relations:

select name, instructor.dept_ name, building from instructor, department where instructor.dept_ name= department.dept_ name;

The Natural Join

- The matching condition in the from clause most often requires all attributes with matching names to be equated.
- This can be done easier in SQL with the help of **natural join**.
- In SQL information from one or two relations can be joined together along with **where** clause.
- The natural join operation operates on two relations and produces a relation as the result.
- It considers only those pairs of tuples with the same value on those attributes that appear in the schemas of both relations.

Queries

For all instructors in the university who have taught some course, find their names and the course ID of all courses can be written as:

Using from … where

select name, course_id from instructor, teaches where instructor.ID= teaches.ID;

Using natural join

select name, course_id from instructor natural join teaches;

A from clause in an SQL query can have multiple relations combined using natural join, as shown:

Select $A_1,A_2,\ldots,A_n$ from r1 natural join r2 natural …natural join rm where P;

from clause can be of the form :

from $E_1, E_2,\ldots,E_n$

where E_i can be single relation or an expression involving natural joins.

Additional Basic Operations:

select name, course_id from instructor, teaches where instructor.ID=teaches.ID;

The result of this query is a relation with attributes :

name,course_id

Drawback of **from:**

- Two relations in the ***from*** clause may have attributes with the same name, in which case an attribute name is duplicated in the result.
- If we used an ***arithmetic expression*** in the select clause, the resultant attribute does not have a name.
- Even if an ***attribute name can be derived from the base relations*** as in the preceding example, we may not change the name in the result.
- SQL provides a way of remaining the attributes of a result relation.
- It uses the ***as*** clause, taking the form:

  old_name as new_name

- Select name as instructor_name, course_id from instructor, teaches where instructor.ID=teaches.ID;
- Rename query useful to make long relation name with short name.
- Select T.name,S.course.id from instructor as T, teaches as S where T.ID=S.ID;
- To “find the names of all instructors whose salary is greater than at least one instructor in the Biology department”.

- It can be write as
    - Select distinct T.name from instructor as T, instructor as S where T.salary>S.salary and S.dept_name='Biology';

String Operations

SQL specifies strings by enclosing them in single quotes, for example, 'Computer'.

A single quote character that is part of a string can be specified by using two single quote characters.

for example, the string "It's right" can be specified by "It"s right".

- The SQL standard specifies that the equality operation on strings is case sensitive; as a result the expression "'comp. sci.' = 'Comp. Sci.'" evaluates to false.
- However, some database systems, such as MySQL and SQL Server, do not distinguish uppercase from lowercase when matching strings; as a result "'comp. sci.' = 'Comp. Sci.'" would evaluate to true on these databases.
- Concatenating using "||"
- extracting substrings
- finding the length of strings
- converting strings to uppercase using the function upper(s) where s is a string
- converting strings to lowercase using the function lower(s)
- removing spaces at the end of the string using trim(s)
- Pattern matching can be performed on strings, using the operator like.
- Describe patterns by using two special characters:
- Percent (%):

The % character matches any substring.

- Underscore (_):

The character matches any character

- 'Intro%' matches any string beginning with "Intro".
- '%Comp%' matches any string containing "Comp" as a substring

- '_ _ _ ' matches any string of exactly three characters.
  - '_ _ _ %' matches any string of at least three characters.
- SQL expresses patterns by using the like comparison operator.

  "Find the names of all departments whose building name includes the substring 'Watson'."

  select dept name from department where building like '%Watson%';
- The escape character is used immediately before a special pattern character to indicate that the special pattern character is to be treated like a normal character.
- The escape character for a like comparison using the escape keyword.
- like 'ab\%cd%' escape '\' matches all strings beginning with "ab%cd".
  - like 'ab\\cd%' escape '\' matches all strings beginning with "ab\cd".
- The asterisk symbol " * " can be used in the select clause to denote "all attributes."

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

  indicates that all attributes of instructor are to be selected.
- The order by clause causes the tuples in the result of a query to appear in sorted order.
- To list in alphabetic order all instructors in the Physics department

  select name from instructor where dept name = 'Physics' order by name;
- By default, the order by clause lists items in ascending order.
- To specify the sort order, specify desc for descending order or asc for ascending order.

- list the entire instructor relation in descending order of salary but the instructors have the same salary,
- Then sort them in ascending order by name.

select * from instructor order by salary desc, name asc;

- SQL includes a between comparison operator to simplify where clauses.

The names of instructors with salary amounts between $90,000 and $100,000

select name from instructor where salary between 90000 and 100000;

instead of:

select name from instructor where salary <= 100000 and salary >= 90000;

Set Operations

- The SQL operations union, intersect, and except operate on relations and correspond to the mathematical set-theory operations ∪, ∩, and −.
- The set of all courses taught in the Fall 2009 semester:
  - ***select course _id from section where semester = 'Fall' and year= 2009;***
- The set of all courses taught in the Spring 2010 semester:

select course _id from section where semester = 'Spring' and year= 2010;

The Union Operation

- To find the set of all courses taught either in Fall 2009 or in Spring 2010, or both,
- **(select course_id from section where semester = 'Fall' and year= 2009) union (select course_id from section where semester ='Spring' and year= 2010);**
- The union operation automatically eliminates duplicates, unlike the select clause.
- If we want to retain all duplicates, we must write union all in place of union:

(select course_id from section where semester = 'Fall' and year= 2009) union all (select course_id from section where semester = 'Spring' and year= 2010);

The Intersection Operation

- To find the set of all courses taught in the Fall 2009 as well as in Spring 2010.

(select course id from section where semester = 'Fall' and year= 2009) intersect (select course id from section where semester = 'Spring' and year= 2010);

- The intersect operation automatically eliminates duplicates.
- If we want to retain all duplicates, we must write intersect all in place of intersect.

(select course id from section where semester = 'Fall' and year= 2009) intersect all (select course id from section where semester = 'Spring' and year= 2010);

The Except Operation

- To find all courses taught in the Fall 2009 semester but not in the Spring 2010 semester,

(select course id from section where semester = 'Fall' and year= 2009) except (select course id from section where semester = 'Spring' and year= 2010);

- The except operation outputs all tuples from its first input that do not occur in the second input; that is, it performs set difference.
- If we want to retain duplicates, we must write except all in place of except

(select course id from section where semester = 'Fall' and year= 2009) except all (select course id from section where semester = 'Spring' and year= 2010);

Null Values

- Null values present special problems in relational operations, including arithmetic operations, comparison operations, and set operations.

Aggregate Functions

In database management, an **aggregate function** or **aggregation function** is a function where the values of multiple rows are grouped together to form a single summary value.

Common aggregate functions include:

- Average (i.e., arithmetic mean)
- Count
- Maximum
- Minimum
- Sum

Average:

select avg (salary) from instructor where dept name= 'Comp. Sci.';

Count:

select count (distinct ID) from teaches where semester = 'Spring' and year = 2010;

The Having Clause:

select dept name, avg (salary) as avg salary from instructor group by dept name having avg (salary) > 42000;

select course id, semester, year, sec id, avg (tot cred) from takes natural join student where year = 2009 group by course id, semester, year, sec id having count (ID) >= 2;

Aggregation by grouping

select dept name, avg (salary) as avg salary from instructor group by dept name;

Nested Queries

The in connective tests for set membership, where the set is a collection of values produced by a select clause. The not in connective tests for the absence of set membership.

select distinct course id from section where semester = 'Fall' and year= 2009 and course id in (select course id from section where semester = 'Spring' and year= 2010);

select distinct course id from section where semester = 'Fall' and year= 2009 and course id not in (select course id from section where semester = 'Spring' and year= 2010);

Set Comparision

select distinct T.name from instructor as T, instructor as S where T.salary > S.salary and S.dept name = 'Biology';

select name from instructor where salary > some (select salary from instructor where dept name = 'Biology');

The subquery: **(select salary from instructor where dept name = 'Biology')**

SQL also allows < some, <= some, >= some, = some, and <> some comparisons.

select name from instructor where salary > all (select salary from instructor where dept name = 'Biology');

The With Clause

The with clause provides a way of defining a temporary relation whose definition is available only to the query in which the with clause occurs.

with max budget (value) as (select max(budget) from department) select budget from department, max budget where department.budget = max budget.value;

with dept total (dept name, value) as (select dept name, sum(salary) from instructor group by dept name), dept total avg(value) as (select avg(value) from dept total) select dept name from dept total, dept total avg where dept total.value >= dept total avg.value;

Modification of the Database

Deletion:

A delete request is expressed in much the same way as a query. It delete only whole tuples and cannot delete values on only particular attributes. SQL expresses a deletion by delete from r where P; where P represents a predicate and r represents a relation. The delete statement first finds all tuples t in r for which P(t) is true, and then deletes them from r.

Insertion:

To insert data into a relation, a tuple can be inserted or write a query whose result is a set of tuples to be inserted. Obviously, the attribute values for inserted tuples must be members of the corresponding attribute's domain. Similarly, tuples inserted must have the correct number of attributes.

insert into course values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);

In certain situations, we may wish to change a value in a tuple without changing all values in the tuple. For this purpose, the update statement can be used.

Updates:

In certain situations, we may wish to change a value in a tuple without changing all values in the tuple. For this purpose, the update statement can be used.

update instructor set salary= salary * 1.05;

SQL provides a case construct that we can use to perform both the updates with a single update statement, avoiding the problem with the order of updates.

update instructor set salary = case

when salary <= 100000

then salary * 1.05 else salary * 1.03

end

The general form of the case statement is as follows.

case

when pred1 then result1 when pred2 then result2 . . . when predn then resultn else result0

end

Join Expressions:

Different Types of SQL JOINs:

- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Returns all records from the left table, and the matched records from the right table

- **RIGHT (OUTER) JOIN**: Returns all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Returns all records when there is a match in either left or right table

INNER JOIN: The INNER JOIN keyword selects all rows from both the tables as long as the condition satisfies. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be same.

Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
INNER JOIN table2
ON table1.matching_column = table2.matching_column;
```

LEFT JOIN: This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of join. The rows for which there is no matching row on right side, the result-set will contain *null*. LEFT JOIN is also known as LEFT OUTER JOIN.

Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

LEFT JOIN table2

ON table1.matching_column = table2.matching_column;
```

RIGHT JOIN: RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of join. The rows for which there is no matching row on left side, the result-set will contain *null*. RIGHT JOIN is also known as RIGHT OUTER JOIN.

Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

RIGHT JOIN table2

ON table1.matching_column = table2.matching_column;
```

FULL JOIN: FULL JOIN creates the result-set by combining result of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both the

tables. The rows for which there is no matching, the result-set will contain *NULL* values.

Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
FULL JOIN table2
ON table1.matching_column = table2.matching_column;
```

Views:

A View can be created from a single table or multiple tables.

Syntax:

```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE condition;
```

In this example, we will create a view named StudentNames from the table StudentDetails.

Query:

```
CREATE VIEW StudentNames AS
SELECT S_ID, NAME
FROM StudentDetails
ORDER BY NAME;
```

If we now query the view as,

```
SELECT * FROM StudentNames;
```

Creating view from multiple tables:

```
CREATE VIEW MarksView AS
SELECT StudentDetails.NAME, StudentDetails.ADDRESS,
StudentMarks.MARKS
FROM StudentDetails, StudentMarks
WHERE StudentDetails.NAME = StudentMarks.NAME;
```

To display data of View MarksView:

```
SELECT * FROM MarksView;
```

Updating Views:

There are certain conditions needed to be satisfied to update a view. If any one of these conditions is **not** met, then we will not be allowed to update the view.

1. The SELECT statement which is used to create the view should not include GROUP BY clause or ORDER BY clause.
2. The SELECT statement should not have the DISTINCT keyword.
3. The View should have all NOT NULL values.
4. The view should not be created using nested queries or complex queries.
5. The view should be created from a single table. If the view is created using multiple tables then we will not be allowed to update the view.

- We can use the **CREATE OR REPLACE VIEW** statement to add or remove fields from a view.

  Syntax:

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1,coulmn2,..
FROM table_name
WHERE condition;
```

Transactions:

Transactions group a set of tasks into a single execution unit. Each transaction begins with a specific task and ends when all the tasks in the group successfully complete. If any of the tasks fail, the transaction fails. Therefore, a transaction has only two results: **success** or **failure**.

Incomplete steps result in the failure of the transaction. A database transaction, by definition, must be atomic, consistent, isolated and durable. These are popularly known as

ACID properties.

COMMIT:

If everything is in order with all statements within a single transaction, all changes are recorded together in the database is called **committed**. The COMMIT command saves all the transactions to the database since the last COMMIT or ROLLBACK command.

ROLLBACK:

If any error occurs with any of the SQL grouped statements, all changes need to be aborted. The process of reversing changes is called **rollback**. This command can

only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.

Integrity Constraints:

Constraints are the rules that we can apply on the type of data in a table. That is, we can specify the limit on the type of data that can be stored in a particular column in a table using constraints.The available constraints in SQL are:

- **NOT NULL**: This constraint tells that we cannot store a null value in a column. That is, if a column is specified as NOT NULL then we will not be able to store null in this particular column any more.
- **UNIQUE**: This constraint when specified with a column, tells that all the values in the column must be unique. That is, the values in any row of a column must not be repeated.
- **PRIMARY KEY**: A primary key is a field which can uniquely identify each row in a table. And this constraint is used to specify a field in a table as primary key.
- **FOREIGN KEY**: A Foreign key is a field which can uniquely identify each row in a another table. And this constraint is used to specify a field as Foreign key.
- **CHECK**: This constraint helps to validate the values of a column to meet a particular condition. That is, it helps to ensure that the value stored in a column meets a specific condition.
- **DEFAULT**: This constraint specifies a default value for the column when no value is specified by the user.

SQL Data types and Schemas:**SQL Date and Time Data Types**

Datatype	Description
DATE	Stores date in the format YYYY-MM-DD
TIME	Stores time in the format HH:MI:SS
DATETIME	Stores date and time information in the format YYYY-MM-DD HH:MI:SS
TIMESTAMP	Stores number of seconds passed since the Unix epoch (‘1970-01-01 00:00:00’ UTC)

YEAR	Stores year in 2 digit or 4 digit format. Range 1901 to 2155 in 4-digit format. Range 70 to 69, representing 1970 to 2069.

SQL Character and String Data Types

Datatype	Description
CHAR	Fixed length with maximum length of 8,000 characters
VARCHAR	Variable length storage with maximum length of 8,000 characters
VARCHAR(max)	Variable length storage with provided max characters, not supported in MySQL
TEXT	Variable length storage with maximum size of 2GB da

SQL Miscellaneous Data Types

Datatype	Description
CLOB	Character large objets that can hold up to 2GB
BLOB	For binary large objects
XML	for storing xml data
JSON	for storing JSON data

Schemas, Catalogs and Environments:

Contemporary database systems provide a three-level hierarchy for naming relations. The top level of the hierarchy consists of catalogs, each of which can contain schemas. SQL objects such as relations and views are contained within a

schema. (Some database implementations use the term “database” in place of the term catalog.)

In order to perform any actions on a database, a user (or a program) must first connect to the database. The user must provide the user name and usually, a password for verifying the identity of the user. Each user has a default catalog and schema, and the combination is unique to the user. When a user connects to a database system, the default catalog and schema are set up for the connection; this corresponds to the current directory being set to the user’s home directory when the user logs into an operating system. To identify a relation uniquely, a three-part name may be used,

for example,

catalog5.univ schema.course

The default catalog and schema are part of an SQL environment that is set up for each connection. The environment additionally contains the user identifier (also referred to as the authorization identifier). All the usual SQL statements, including the DDL and DML statements, operate in the context of a schema.

Authorization:

Authorizations on data include:

- Authorization to read data.
- Authorization to insert new data.
- Authorization to update data.
- Authorization to delete data.

Granting and Revoking of privileges:

The SQL standard includes the privileges **select, insert, update and delete** .

The privilege **all privileges** can be used as a short form for all the allowable privileges. A user who creates a new relation is given all privileges on that relation automatically.

The grant statement is used to confer authorization.

grant<privilege list>

on <relation name or view name>

to <user/role list>;

Example:

grant select on department to Amit, Santhosh;

grant update(budget) on department to Amit, Santhosh;

To revoke an authorization:

revoke<privilege list> on <relation name or view name> from <user/role list>;

revoke select on department from Amit, Santhosh;

revoke update(budget) on department from Amit, Santhosh;

Roles:

A set of roles is created in the database. Each database user is granted a set of roles. Roles can be created as

create role instructor;

Roles can then be granted privileges as:

grant select on takes **to** instructor;
Authorization on views:

create view geo_instructor as (select * from instructor where dept_name='Geology');

Then it can answer for the query

select * from geo_instructor;

Authorizations on schema:

grant references(dept_name) **on** department **to** Mariano;

Transfer of Privileges:

grant select on department **to** Amit **with grant option**;

Revoking of Privileges:

revoke select on department **from** Amit, Santhosh **restrict**;

revoke grant option for select on department **from** Amit;

UNIT-IV

DATABASE DESIGN AND THE E-R MODEL

OVERVIEW OF DESIGN PROCESS:

Creating a database involve

- DESIGN OF THE DATABASE SCHEMA
- DESIGN OF THE PROGRAMS THAT ACCESS,AND UPDATE THE DATA
- DESIGN OF A SECURITY SCHEME TO CONTROL ACCESS TO DATA

DESIGN PHASE

Requirements Analysis:

- Understand the requirements related to the relation, their attributes and constraints on the relation
- Designer interact with domain experts and users to carry out this characterization of data requirements.

Conceptual Design Phase:

- The designer chooses a data model and applying the concepts of the chosen model and translates these requirements into conceptual schema.
- The schema developed at this conceptual design phase provides a detailed overview of the enterprise.
- The Entity-Relationship model stated the conceptual design specifies the entities in the database, the attributes of the entities, the relationships among the entities and constraints on the entities and relationships

Conceptual Design Phase:

- The designer chooses a data model and applying the concepts of the chosen model and translates these requirements into conceptual schema

- The schema developed at this conceptual design phase provides a detailed overview of the enterprise.
- The Entity-Relationship model stated the conceptual design specifies the entities in the database, the attributes of the entities, the relationships among the entities and constraints on the entities and relationships.

Functional Requirements:

- A fully developed conceptual schema indicates the functional requirements of the enterprise.
- In a Specification of functional requirement, users describe the kinds of operations that will be performed on the data.
- Operations include modifying, updating data, searching for and retrieving specific data a d deleting data

Design Alternatives:

Major pitfalls in designing database:

Redundancy and Incompleteness

- **Redundancy**: A bad design may repeat information.
  - ✓ Data **redundancy** occurs when the same piece of data is stored in two or more separate places. May result in inconsistent database.For example different offerings of a course may have same course identifier, but may have different titles. It would then become unclear what the correct title of the course is. Idealy, information should appear in exactly one place.
  - ✓ **Incompleteness**: Suppose we have a single relation where we repeat all of the course information once for each section that the course is offered. It would then be impossible to represent information about a new course ,unless that course is offered. Such a model is unattractive ,but may be prevented by a primary key constraints

The Entity-Relation Model

- The E-R model is useful in mapping the meanings and interactions of real-world enterprises onto a conceptual schema.
- **Entity Set:** An entity is **'thing' or 'object'** .An entity had a set of attributes. The values for some set of properties may uniquely identify the entity.
- An entity set is a set of entities of same type that share the same properties, or **attributes.**
- Entity set do not need to be **disjoint**. An entity is represented by a set of attributes. Attributes are descriptive properties possessed by each member of an entity set.

  Each entity has a value for each of its attributes.

Relationship set: A relationship is an association among several entities.

- A relationship set is a ser of relationships of the same type. It is a mathematical relation on n>=2 entity sets.
- If E1,E2,...En are entity sets, then a relationship set R is a subset of (e1,e2,...en)|e1€ E1,E2,... €En, where (e1,e2..en) is a relationship.
- The association between entity sets is referred to as participation. The entity sets E1,E2...En participate in relationship R.
- A relationship **instance** in an E-R schema represents an association between the named entities in the real world enterprise that is being modeled.
- The function that an entity plays in a relationship is called that entity's **role.**
- Same entity set participates in a relationship set more than once, in different roles called a **recursive** relationship set.
- A relationship may also have attributes called descriptive attributes.

- A **relationship instance** in a given relationship set must be uniquely identifiable from its participating entities, without using the descriptive attributes.

  Attributes: For each attribute, there is a set of permitted values, called the Domain or value set of that attribute.

Attribute types in E-R Model:

- ✓ ***Simple Attributes***: Attributes that have not been divided into subparts.
- ✓ **CompositeAttributes:** Compostie Attributes are made of more than one simple attributes. For,example name attribute can be divided into first-name,last-name ,middle-name.
- ✓ **Single-valued and multivalued Attribute**: Attributes which have single value for each entity.(Student-ID). There may be instances where an attribute has set of values for specific entity called **multivalued arrtibutes.**
- ✓ **Derived Attribute:** The value for this entity can be derived from the values of other related attributes or entities. Example , Instructor entity set has an entity *age and date- of-birth* entity, From *date-of-birth* we can derive his age.

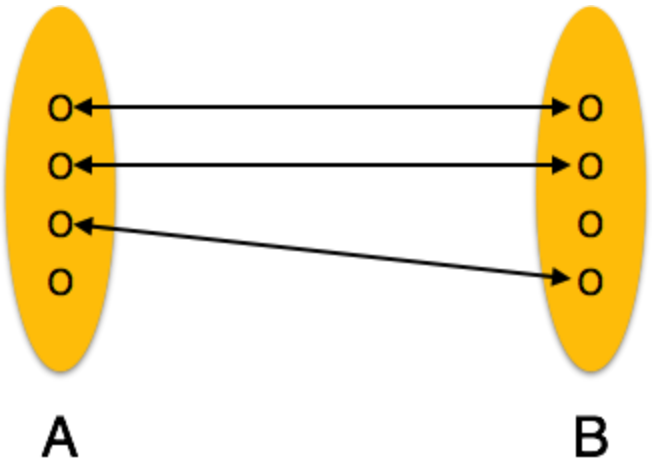
Constraints:

- **Mapping Cardinality**: Cardinalities defines the number of entities in one entity set , which can be associated with the number of entities of other set via relationship set.

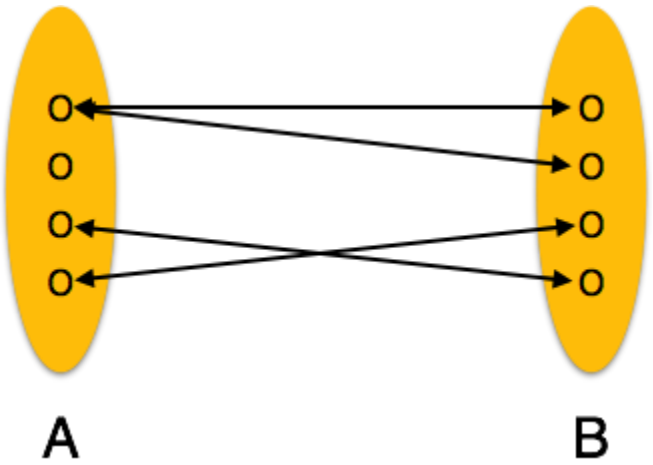
Mapping Cardinalities

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

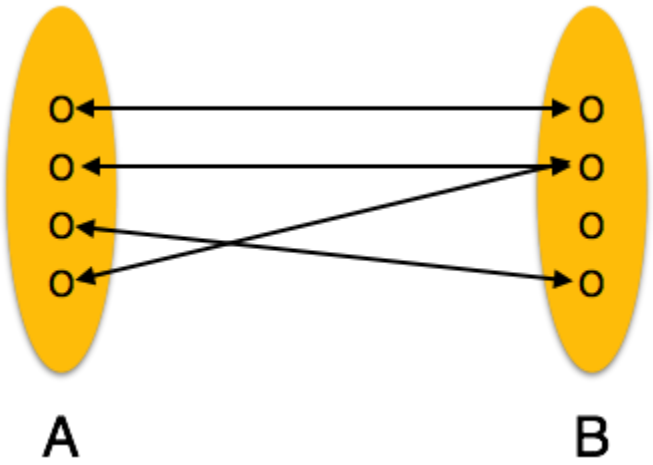
- **One-to-one** − One entity from entity set A can be associated with at most one entity of entity set B and vice versa.



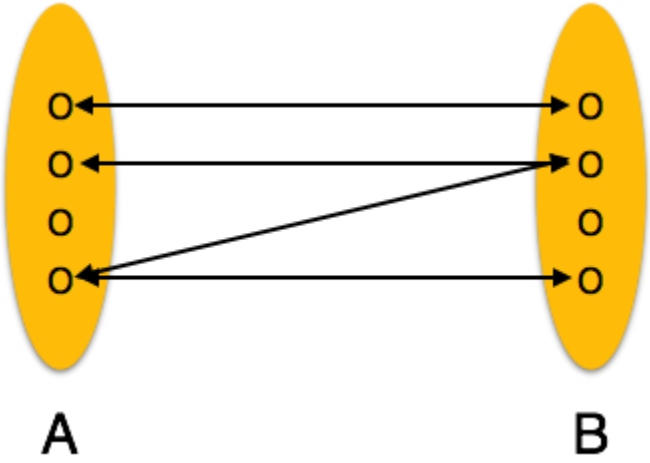
- **One-to-many** − One entity from entity set A can be associated with more than one entities of entity set B however an entity from entity set B, can be associated with at most one entity.



- **Many-to-one** − More than one entities from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A.



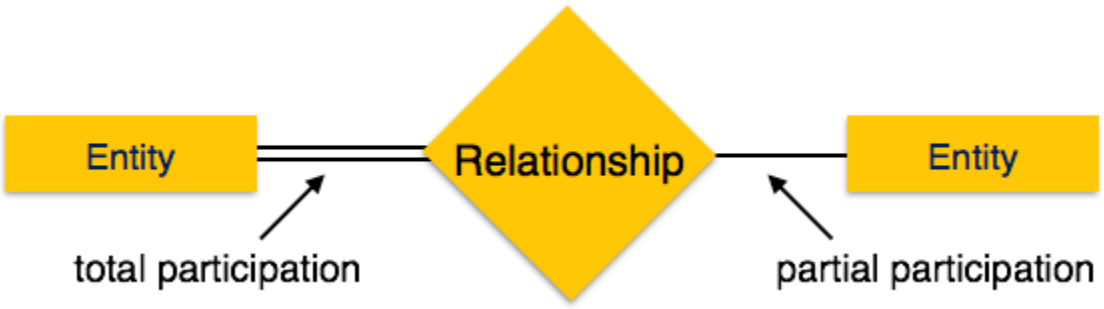
- **Many-to-many** − One entity from A can be associated with more than one entity from B and vice versa.



Participation Constraints:

Participation Constraints

- **Total Participation** − Each 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 lines.



Keys:

- Key for an entity is a set of attributes that suffice to distinguish entities from each other. Keys uniquely identify the entity.
- Primary key of an entity set allows us to distinguish among the various entities of the set.

- Let R be a relationship set involving entity sets E1,E2,...En.Let Primarykey(Ei) denote the set of attributes that forms the primary ket for entity set Ei.

Removing Redundant Attributes in Entity Set:

A good entity-relationshop design does not contain reduntant attributes.

- Once the entities and the corresponding attributes are chosen, the relationship sets among the various entities are formed.
- These relationship sets may result in a situation where attributes in the various entity sets are redundant and need to be removed from the original entity sets.

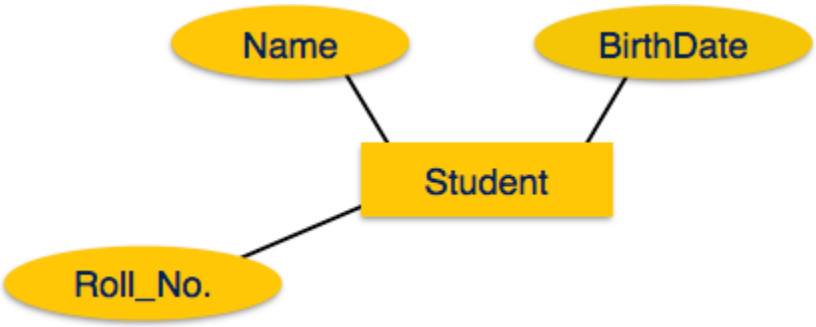
Entity –Relationship Diagrams

It is the logical structure of a database.g Entity

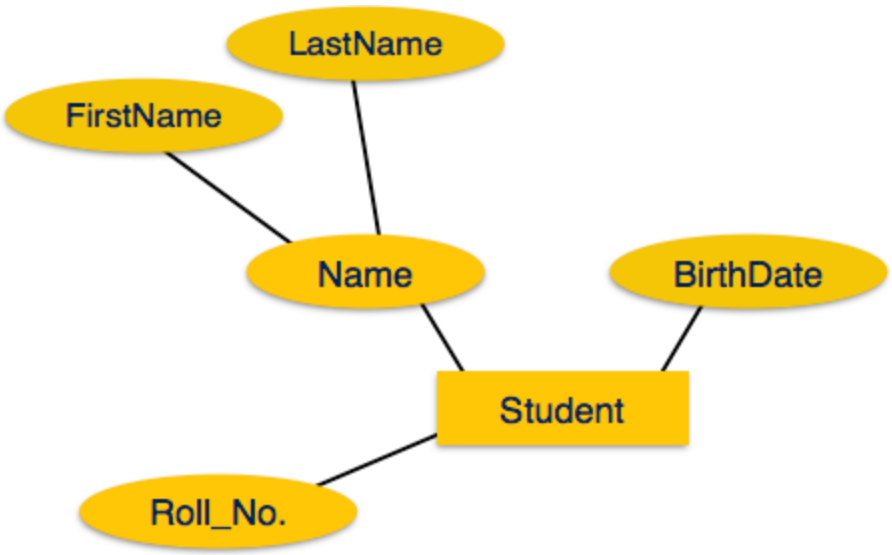
Entities are represented by means of rectangles. Rectangles are named with the entity set they represent.

Attributes

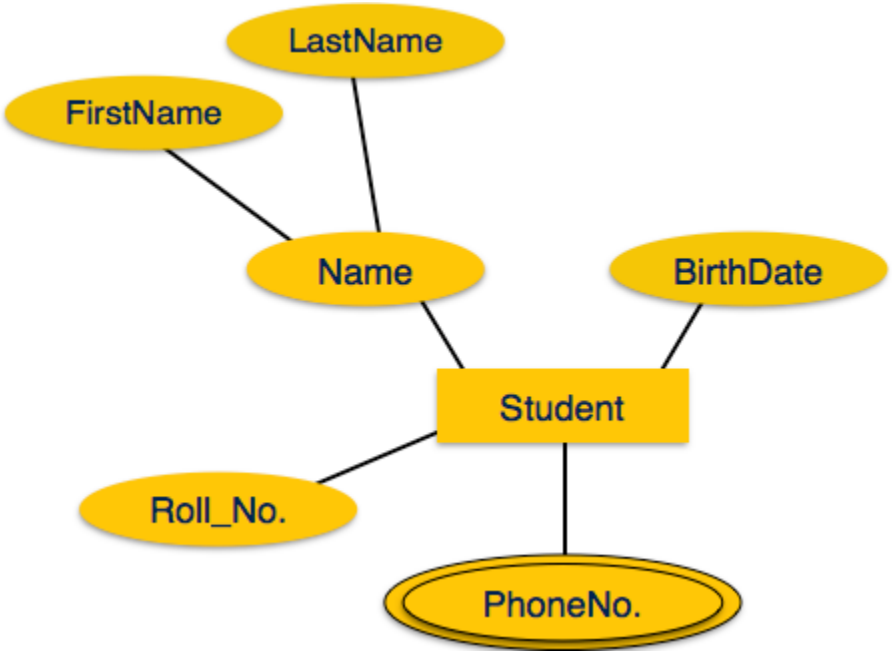
Attributes are the properties of entities. Attributes are represented by means of ellipses. Every ellipse represents one attribute and is directly connected to its entity (rectangle).



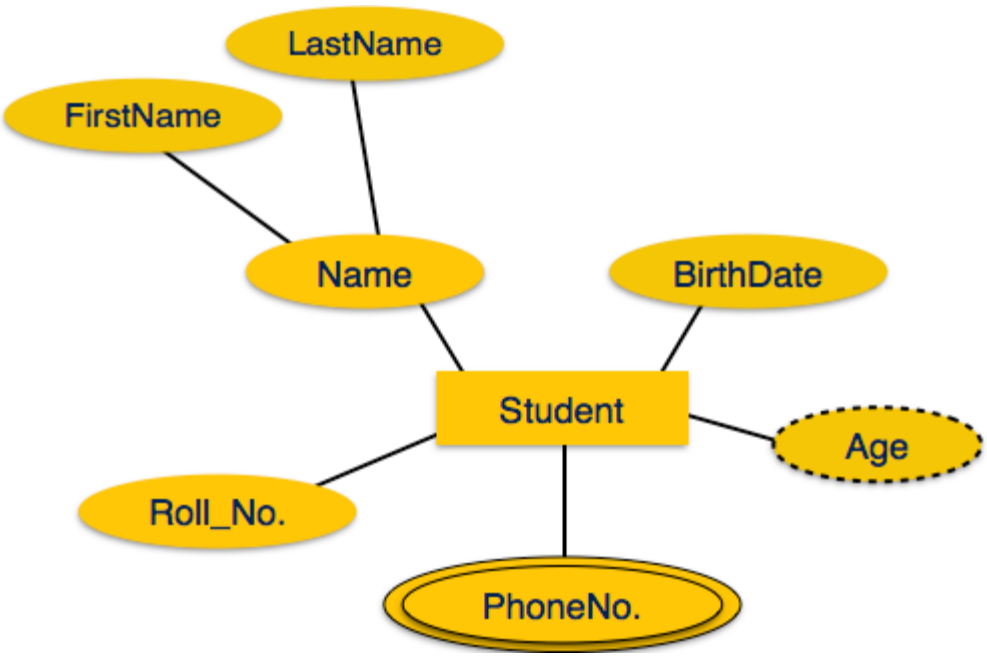
If the attributes are **composite**, they are further divided in a tree like structure. Every node is then connected to its attribute. That is, composite attributes are represented by ellipses that are connected with an ellipse.



Multivalued attributes are depicted by double ellipse.

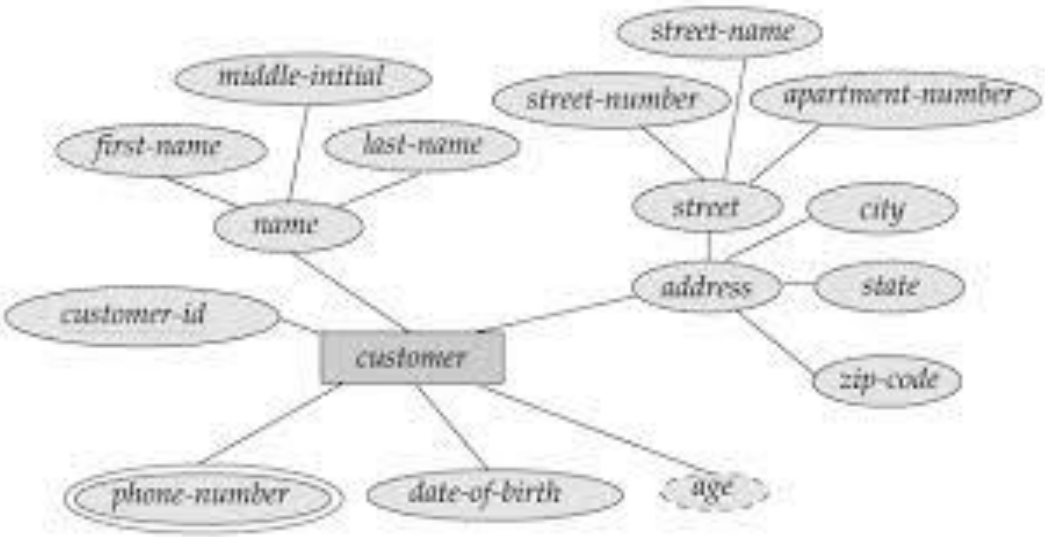


Derived attributes are depicted by dashed ellipse.



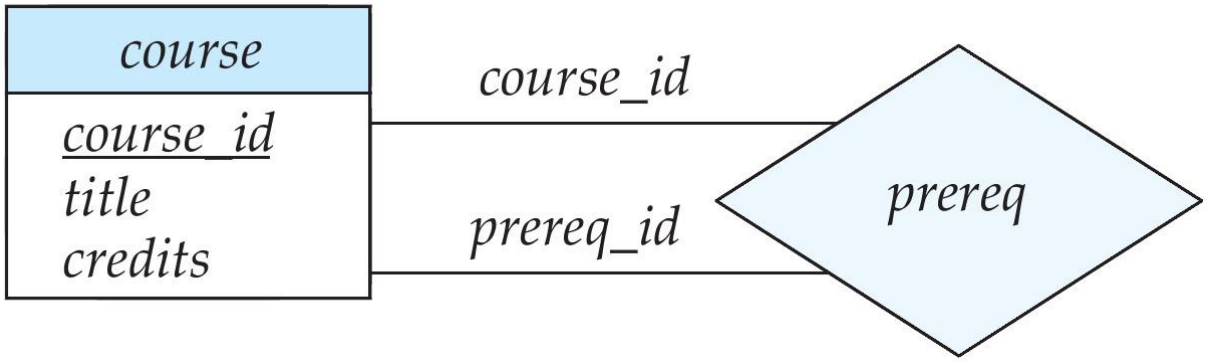
Complex attributes:

The **attributes** that are formed by arbitrarily nesting the **composite** and multivalued **attributes** are called **complex attributes**.

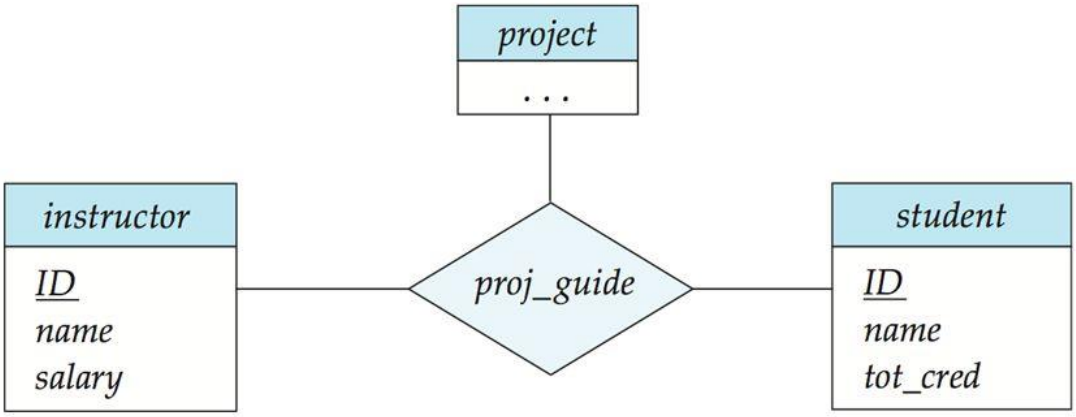


Roles:

Role indicaed by labeling the lines that connect diamonds to rectangle

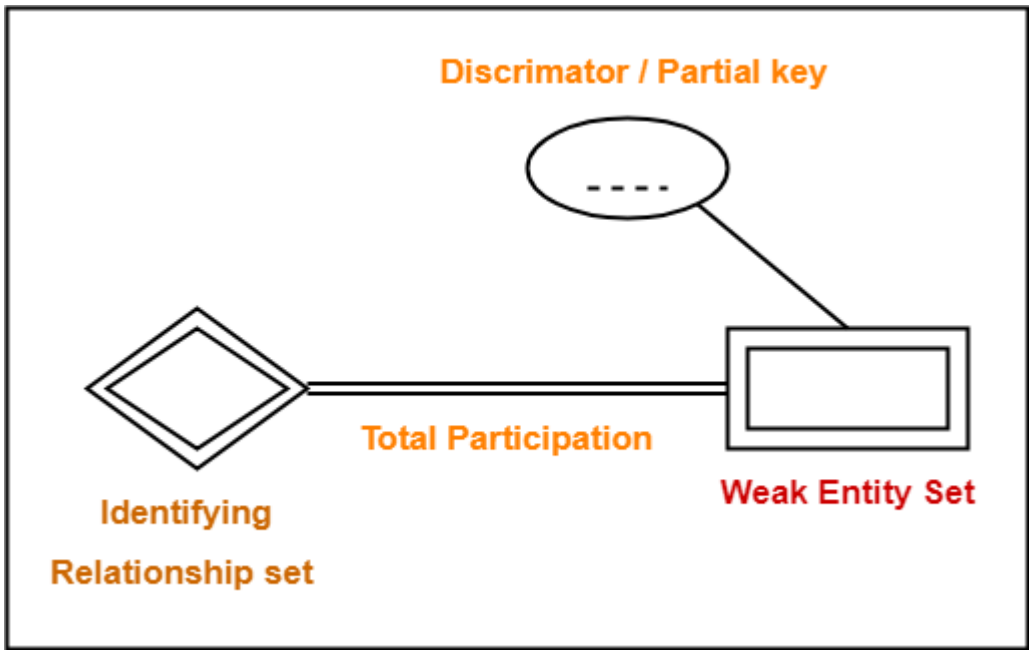


Non binary relationships:



Weak Entity Set

- ✓ A **weak entity** always has a total participation constraint with respect to its identifying relationship because it cannot be identified independently of its owner identity. A **weak entity** may have a partial key, which is a list of attributes that identify **weak entities** related to the same owner **entity**.
- ✓ The entity sets which do not have sufficient attributes to form a primary key are known as **weak entity sets** and the entity sets which have a primary key are known as strong entity sets.
- ✓ Weak entity is **depend on strong entity** to ensure the existence of weak entity. Like strong entity weak entity does not have any primary key, It has partial discriminator key.
- ✓ Weak entity is represented by double rectangle. The relation between one strong and one weak entity is represented by **double diamond**. **Partial Key** attributes are represented with **dotted lines.**
- ✓ As the weak entities do not have any primary key, they cannot be identified on their own, so they depend on some other entity (known as owner entity).
- ✓ The weak entities have total participation constraint (existence dependency) in its identifying relationship with owner identity.
- ✓ Weak entity types have partial keys. Partial Keys are set of attributes with the help of which the tuples of the weak entities can be distinguished and identified.



RELATIONAL DATABASE DESIGN

Database Design Objective

- **Eliminate Data Redundancy:** the same piece of data shall not be stored in more than one place. This is because duplicate data not only waste storage spaces but also easily lead to inconsistencies.
- **Ensure Data Integrity and Accuracy:** is the maintenance of, and the assurance of the accuracy and consistency of, data over its entire life-cycle, and is a critical aspect to the design, implementation, and usage of any system which stores, processes, or retrieves data.
- A relation schema might contain certain dependencies in whihcase t should decomposed(normalized) into multiple smaller relation schemas.
- This normalization process is based on functional dependencies multivalued dependencies.

Features of Good Relational Designs:

Design Alternatives in Larger Schema:

Design Alternatives in Smaller Schema:

- ✓ Suppose we had started with inst_deptp and decompose) it into instructor and department
- ✓ Write a rule "**if there were a schema (dept_name, building, budget)**, then dept_name would be a candidate key"

Denote as a **functional dependency**:

$$\text{building, budget} \rightarrow \text{dept\_name}$$

- ✓ In inst_dept, because dept_name is not a candidate key, the building and budget of a department may have to be repeated. This indicates the need to decompose inst_dept . This indicates the need to decompose inst_dept , Not all decompositions are good.

Suppose we decompose **employee(ID, name, street, city, salary)** into

employee1 (ID, name)

employee2 (name, street, city, salary)

we cannot reconstruct the original employee relation -- and so, this is a lossy decomposition**.**

Atomic Domains and First Normal Forms:

A ***domain*** is the original sets of atomic values used to model data. By *atomic value*, we mean that each value in the domain is indivisible as far as the relational model is concerned.

For example:

The domain of Marital Status has a set of possibilities: Married, Single, Divorced.

The domain of Shift has the set of all possible days: {Mon, Tue, Wed…}.

The domain of Salary is the set of all floating-point numbers greater than 0 and less than 200,000.

The domain of First Name is the set of character strings that represents names of people.

- ✓ A relational schema R is in first normal form if the domains of all attributes of R are atomic

- ✓ Non-atomic values complicate storage and encourage redundant (repeated) storage of data Atomicity is actually a property of how the elements of the domain are used.

  Example:

  - Strings would normally be considered indivisible

- Suppose that students are given roll numbers which are strings of the form CS0012 or EE1127

- If the first two characters are extracted to find the department, the domain of roll numbers is not atomic.

Decomposition Using Functional Dependencies:

- When a set of attributes is a Super key – denoted by "K".A super key pertains a specific relation schema, " K is a super key of r(R)"
- Relations are denoted as "r"
- A relation has a particular value at any given time, called as the "instance of r"

Keys and Functional Dependency:

Let R be a relation schema $\beta \subseteq$ Rand $\alpha \subseteq$R.

The **functional dependency** $\alpha \rightarrow \beta$ holds on R if and only if for any legal relations r(R), whenever any , they also agree α two tuples t1 and t2 of r agree on the attributes β . That is, t1 [α] = t2 [α]=>t1 [β] = t2 [β]

Let R be a relation schema $\alpha \subseteq$ R and $\beta \subseteq$ R

- ✓ The functional dependency $\alpha \rightarrow \beta$ holds on R if and only if for any legal relations r(R), whenever any , two tuples t1 and t2 of r agree on the attributes α they also agree on the attributes α. That is,

  t1 [α] = t2 [β]$\Rightarrow$t1 [β]=t2 [β]

- ✓ Functional dependencies allow us to express constraints that cannot be expressed using superkeys. Consider the schema:

  inst_dept (ID, name, salary, dept_name, building, budget).

We expect these functional dependencies to hold: building$\rightarrow$dept_name , building and ID but would not expect the following to hold: salary$\rightarrow$dept_name

- ✓ A functional dependency is trivial if it is satisfied by all instances of a relation

  Example: ID$\rightarrow$ID, name

  name$\rightarrow$name

In general, $\alpha \subseteq \beta$ is trivial if $\beta \rightarrow \alpha$

- ✓ Given a set F of functional dependencies, there are certain other functional dependencies that are logically implied by F,

  For example:If A->B and B->C, then we can infer that A->C

- ✓ The set of all functional dependencies logically implied by F is the closure of F. We denote the closure of F by F+ . F+ is a superset of F.

Boyce–Codd normal form

It is a normal form used in database normalization. It is a slightly stronger version of the third normal form (3NF).

If a relational schema is in **BCNF** then all redundancy based on functional dependency has been removed, although other types of redundancy may still exist

A relational schema *R* is in **Boyce–Codd** normal form if and only if for every one of its dependencies $X \rightarrow Y$, at least one of the following conditions hold:

- $X \rightarrow Y$ is a trivial functional dependency ($Y \subseteq X$),
- *X* is a super key for schema *R*.

BCNF and Dependency Preservation:

- ✓

  If a relation is in BCNF it is in 3NF . Constraints, including functional dependencies, are costly to check in practice unless they pertain to only one relation .Because it is not always possible to achieve both BCNF and dependency preservation, we usually consider normally third normal form.

It is always possible to decompose a relation into a set of relations that are in 3NF such that:

- the decomposition is lossless
- the dependencies are preserved

It is always possible to decompose a relation into a set of relations that are in BCNF such that:

- the decomposition is lossless
- it may not be possible to preserve dependencies

Third Normal Form:

- ✓ **Third normal form** (**3NF**) is a database schema design approach for relational databases which uses normalizing principles to reduce the duplication of data, avoid data anomalies, ensure referential integrity, and simplify data management.

Codd's definition states that a table is in 3NF if and only if both of the following conditions hold:

- The relation R (table) is in second normal form (2NF).
- Every non-prime attribute of R is non-transitively dependent on every key of R.

UNIT V

The Cooperative Approach

- Oracle is an object-relational database.
- A relational database is an extremely simple way of thinking about and managing the data used in a business.
- It is nothing more than a collection of tables of data.
- These are all tables, with column headings and rows of information simply presented.
- An object-relational database supports all of the features of a relational database while
- A common language is needed to make this cooperative work.

The familiar Language of Oracle

The information stored in Oracle is kept in tables.

The major characteristics of most tables are columns, rows and a name.

Table of baby-name data
(baby-2010.csv)

name	rank	gender	year
Jacob	1	boy	2010
Isabella	1	girl	2010
Ethan	2	boy	2010
Sophia	2	girl	2010
Michael	3	boy	2010

Field names

One row (4 fields)

2000 rows all told

Codes, Abbreviations and Naming Standards

The problem of old programming habits is most pronounced in codes, abbreviations and naming standards, which are almost completely ignored when the needs of end users are considered. When these three issues are thought about at all, usually only the needs and conventions of the systems groups are considered. Each event or transactionnnnn was written down, line by line, in English.

This problem has been most pronounced in large, conventional mainframe systems development.

Normalizing Names

- Building a logical map between the common English words and the hard to remember, non-English column names, table names and codes.
- The mapping takes careful thought, but once completed, it makes the user's interaction with the application easy.
- For performance reasons, it may be that some of an application's data must still be stored in a coded fashion within the computer's database.
- These codes should not be exposed to users, either during data entry or retrieval and oracle allows them to be easily hidden.
- The instant that data entry requires codes, key-entry errors increase.
- When reports contain codes instead of English, errors of interpretation begin.
- And when users need to create new or ad hoc reports, their ability to do so quickly and accurately is severly impaired both by codes and by not being able to remember strange column and table names.

Understanding the Data

In conjunction with the decomposition and description of the tasks, the resources required at each step are described in the task document, especially in terms of the data required.

This is done on a task-by-task basis, and the data requirements are then included in the data document.

By requiring that the definition of the data needed come from the task rather than from any existing forms or screens.

Once the current data elements have been identified, they must be carefully scrutinized.

Numeric and letter codes are always suspect.

They disguise real information behind counterintuitive , meaningless symbols.

There are times and tasks for which codes are handy , easily remembered, or made necessary by sheer volume.

But, in your final design these cases should be rate and obvious.

In the scrutiny of existing data elements, codes should be set aside for special attention.

In each case ask yourself whether the element should be a code.

Its compelling reasons for perpetuating the viewed suspiciously.

There must be good arguments and compelling reason for perpetuating the disguise.

The process for converting codes back into English is fairly simple, does are first listed, but is a joint effort.

These are them examined by users and designers and short English version of the meaning are proposed, discussed and tentatively approved.

In same discussion designer and end users should decide on names for the data elements.

These will become column names in the database and will be regularly used in English queries, so the names should be descriptive and singular.

To intimate relationship between the column name and the data ir contains the two should be specified simultaneously.

Data element that are not codes also must be rigorously examined.

By the point you have good reason to believe that all of the data elements you have identified are necessary to the business tasks, but they are not necessarily well-organized.

Appears to be one data element in the existing task may in fact be several elements mixed together that require separation.

First and last name were mixed together.

For example, in the AUTHOR table. The authorname column held both first and last names even though the tables were in Third Normal Form.

Extremely burdensome way to actually implements an application in spite of the fact that the normalization rules were technically met.

The application practical and prepare it for English queries the AuthorName column needs to be decomposed into at least two new columns, LastName and FirstName.

The degree of decomposition depends on how the particular data elements are like to be used.

It is possible to much too far and decompose categories though made up of separable pieces, provide no additional value in their new state.

Decomposition is dependent on an element-by-element basis.

These new elements which will become columns, need to thoughtfully named, and the data they will contain needs to be scrutinized.

Text data that will fall into a definable number of values should be reviewed for naming.

Style

First some comments on style. SQLPLUS doesn't care whether the SQL commands you type are in uppercase or lowercase. This command

```
SELECT feature, section, PAGE FROM newspaper;
```

Will produce exactly the same result as this one:

```
select Feature, Section, page form NEWSPAPER;
```

Case matters only when SQLPLUS or oracle is checking an alphanumeric value for equality.

Oracle to find a row where Section='f' and Section is really equal to 'F',Oracle won't find it.

Aside form this use, case is completely irrelevant, the letter ‘F’ as used here, is called a literal, meaning that you want Section to be tested literally against letter ‘F’ not a column named F.

The single quote marks enclosing the letter tell Oracle that this is literal and not a column name.

- **Select, from, where, order by, having** and **group** by will always be lowercase and boldface.
- SQLPLUS commands also will be lowercase and boldface: **column, set, save, ttitle,** and so on.
- **IN, BETWEEN, UPPER** and other SQL operation and function will be uppercase and boldface.
- Column names will be mixed uppercase and lowercase without boldface: Feature, EastWest, Longitude, and so on.
- Tables names will be uppercase without boldface: NEWSPAPER, WEATHER, LOCATION, and so on.

You may want to follow similar convention to creating your own queries, or you company already may have standards, it would like to use, or you may choose to invent your own.

The goal of any such standards should always to be make your work simple to read and understand.

Logic and Value

Just as the **order by** clause can have several parts,so can the **where** clause, but with a significantly greater degree or sophistication.

To control the extend to which you use **where** through the careful use of logical instruction to oracle on what you expect it to return to you.

These instruction are expressed using mathematical symbols called logical operators.

Se are explained shortly and also are listed in the Alphabetical Reference.

The values in the page column are tested to see if any equals 6.

Every row where this is true is returned to you. Any row in which page is not equal to 6 is skipped.

```
select Features, Section, Page

from NEWSPAPER

where Page = 6;
```

FEATURES	S	PAGE
Obituaries	F	6
Doctor Is In	F	6

The equal sign is called a logical operator, it operates by making a logical test that campares the values on either side of it—in this case, the value of page and the value 6—to see if they are equal.

For example no quotes are place around the value being checked, because the column the value is compared is defined as a NUMBER datatype.

Number values do not require quotes around them during comparies.

Combining Logic

Both **AND** and **OR** follow the commonsense meaning of the words. They can be combined in a virtually unlimited number of ways, but you must use care, because **AND** and **OR** get convoluted very easily

You want to find the features in the page that the editors tend to bury, those that are placed somewhere past page 2 of section A or B.

```
select Feature, Section, Page
from NEWSPAPER
where Section = 'A'
or Section = 'B'
and Page = 2;
```

FEATURE	S	PAGE
National News	A	1
Editorials	A	12
Television	B	7
Movies	B	4

The result you got back from Oracle is not what you wanted.

Although both **AND** and **OR** are logical connection, **AND** is stronger.

It binds the logical expression on either side of it more strongly than **OR** does which means this **where** clause.

```
where  Section = 'A'
or Section = 'B'
and Page = 2;
```

interpreted to read, "where Section = 'A' or where Section = 'B' and Page >2".

If you look at each of the failed examples just given, you will this interpretation affected the result. The **AND** is always acted on first.

Break the bonding by using parentheses that enclose those expression you want to be interpreted together. Parentheses override the normal precedence.

```
select Feature, Section, Page
from NEWSPAPER
where Page = 2
and (Section = 'A' or Section = 'B');
```

FEATURE	S	PAGE
Editorials	A	12
Television	B	7
Movies	B	4