DBMS/SQL

Introduction to Database

What is Data?

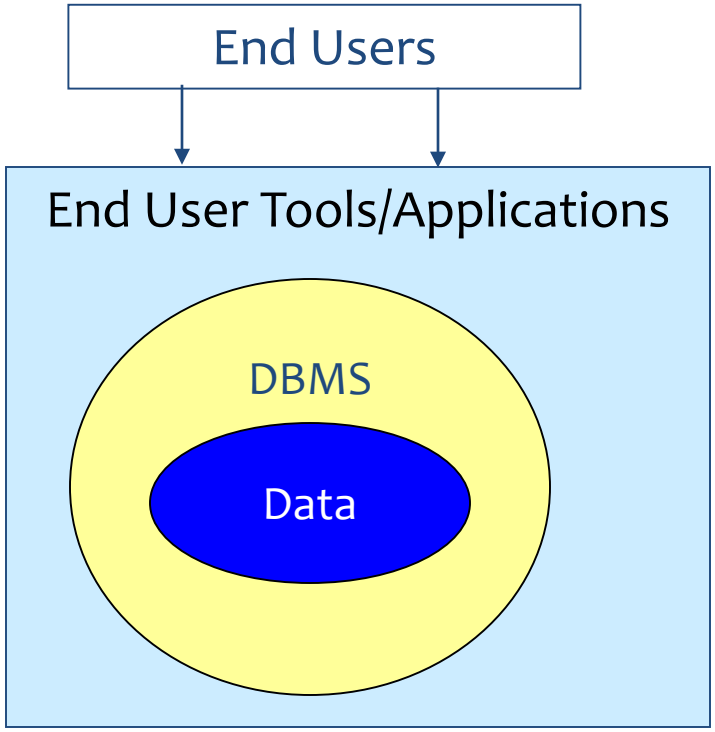
- **Data (plural of the word datum) is a factual information used as a basis for reasoning, discussion, or calculation**
- **Data may be numerical data which may be integers or floating point numbers, and non-numerical data such as characters, date etc.**
- **Data by itself normally doesn't have a meaning associated with it.**
- **e.g:-** Jack
  01-jan-71
  15-jun-05
  50000

What is Information?

- **Related data is called as information**
- **Information will always have a meaning and context attached to the data element**
- **When we add meaning and context to the data it becomes information.**
  - Employee name: Jack
  - Date of birth: 01-jan-71
  - Data of joining: 15-jun-05
  - Salary: 50000
  - Department number: 10

Defining Database, DBMS & Schema

- **Database: It is a set of inter-related data**
- **DBMS: It is a software that manages the data**
- **Schema: It is a set of structures and relationships, which meet a specific need**



Characteristics of DBMS

- **Given below are the characteristics of DBMS:**
  - Control of Data Redundancy
    - Traditionally, same data is stored in a number of places
    - Gives rise to data redundancy and its disadvantages
    - DBMS helps in removing data redundancies by providing means of data-integration.
  - Sharing of Data
    - DBMS allows many applications to share the data.
  - Maintenance of Integrity
    - DBMS maintains the correctness, consistency, and interrelationship of data with respect to the application, which uses the data.

Characteristics of DBMS

- Support for Transaction Control and Recovery
  - DBMS ensures that updates physically take place after a logical Transaction is complete.
- Data Independence
  - In DBMS, the application programs are transparent to the physical organization and access techniques.
- Availability of Productivity Tools
  - Tools like query language, screen and report painter, and other 4GL tools are available.

Characteristics of DBMS

- Control over Security
  - DBMS provides tools with which the DBA can ensure security of the database.
- Hardware Independence
  - Most DBMS are available across hardware platforms and operating systems.

What is a Data Model?

- **The "Data model" defines the range of data structures supported and the availability of data handling languages.**
  - It is a collection of conceptual tools to describe:
    - Data
    - Data relationships
    - Constraints
  - There are different data models:
    - Hierarchical Model
    - Network Model
    - Relational Model

Relational Model

- **The Relational model:**
  - The Relational model developed out of the work done by Dr. E. F. Codd at IBM in the late 1960s. He was looking for ways to solve the problems with the existing models.
  - At the core of the Relational model is the concept of a "table" (also called a "relation"), which stores all data.
  - Each "table" is made up of:
    - "records" (i.e. horizontal rows that are also known as "tuples"), and
    - "fields" (i.e. vertical columns that are also known as "attributes")

Relational Model

- **The Relational model:**
  - Examples of RDBMS:
    - Oracle
    - Informix
    - Sybase
  - Because of lack of linkages, the Relational model is easier to understand and implement.

Student Table	
Scode	Sname
S1	A
S2	B

Course Table	
Ccode	Cname
C1	Physics
C2	Chemistry
C3	Maths
C4	Biology

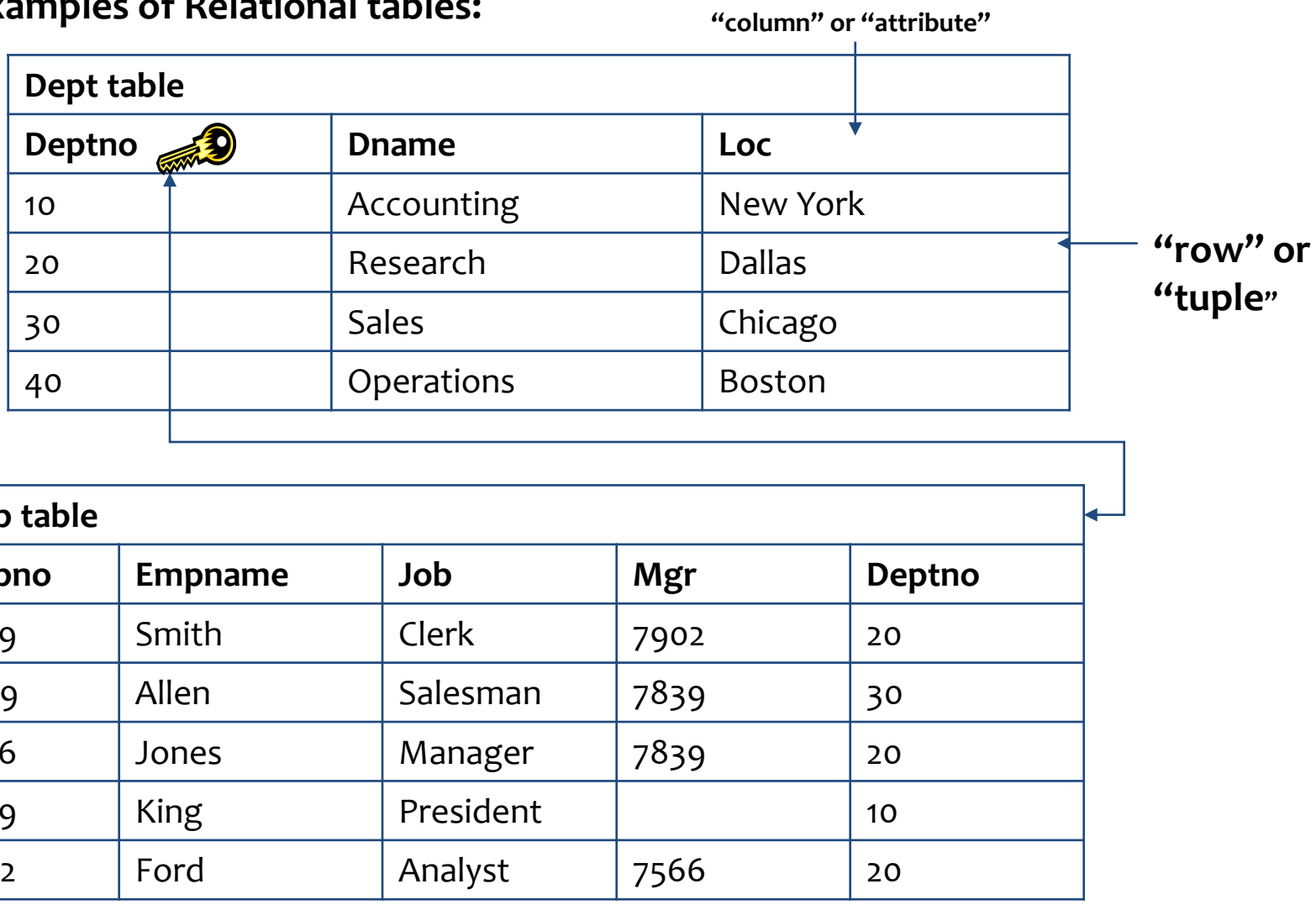
Marks Table		
Ccode	Scode	Marks
C1	S1	65
C2	S1	78
C3	S1	83
C4	S1	85
C3	S2	83
C4	S2	85

Relational Model - Possibilities

- **Possibilities in a Relational model:**
  - INSERT
    - Inserting a “course record” or “student record” poses no problems because tables are separate.
  - UPDATE
    - Update can be done only to a particular table.
  - DELETE
    - Deleting any record affects only a particular table.

Relational Tables

- **Examples of Relational tables:**



Dept table		
Deptno	**Dname**	**Loc**
10	Accounting	New York
20	Research	Dallas
30	Sales	Chicago
40	Operations	Boston

Emp table				
Empno	**Empname**	**Job**	**Mgr**	**Deptno**
7369	Smith	Clerk	7902	20
7499	Allen	Salesman	7839	30
7566	Jones	Manager	7839	20
7839	King	President		10
7902	Ford	Analyst	7566	20

Relational Tables - Properties

- **Properties of Relational Data Entities:**
  - Tables must satisfy the following properties to be classified as relational:
    - Entries of attributes should be single-valued.
    - Entries of attributes should be of the same kind.
    - No two rows should be identical.
    - The order of attributes is unimportant.
    - The order of rows is unimportant.
    - Every column can be uniquely identified.

Data Integrity

- **“Data Integrity” is the assurance that data is consistent, correct, and accessible throughout the database.**
- **Some of the important types of integrities are:**
  - Entity Integrity:
    - It ensures that no “records” are duplicated, and that no “attributes” that make up the primary key are NULL.
    - It is one of the properties that is necessary to ensure the consistency of the database.

Data Integrity

- **Foreign Key and Referential Integrity**
  - The Referential Integrity rule: If a Foreign key in table A refers to the Primary key in table B, then every value of the Foreign key in table A must be null or must be available in table B.
- **Unique Constraint:**
  - It is a single field or combination of fields that uniquely defines a tuple or row.
  - It ensures that every value in the specified key is unique.
  - A table can have any number of unique constraints, with at most one unique constraint defined as a Primary key.
  - A unique constraint can contain NULL value.

Data Integrity

- **Column Constraint:**
  - It specifies restrictions on the values that can be taken by a column.

DEPT table		
Deptno	Dname	Loc
10	Accounting	New York
20	Research	Dallas

EMP table				
Empno	Empname	Job	Mgr	Deptno
7369	Smith	Clerk	7902	20
7499	Allen	Salesman	7839	30

DBMS/SQL

Basics of SQL

What is SQL?

- **SQL:**
  - SQL stands for Structured Query Language.
  - SQL is used to communicate with a database.
  - Statements are used to perform tasks such as update data on a database, or retrieve data from a database.
  - Benefits of SQL are:
    - It is a Non-Procedural Language.
    - It is a language for all users.
    - It is a unified language.

Rules for SQL statements

- **Rules for SQL statements:**
  - SQL keywords are not case sensitive. However, normally all commands (SELECT, UPDATE, etc) are upper-cased.
  - “Variable” and “parameter” names are displayed as lower-case.
  - New-line characters are ignored in SQL.
  - Many DBMS systems terminate SQL statements with a semi-colon character.
  - “Character strings” and “date values” are enclosed in single quotation marks while using them in WHERE clause or otherwise.

Standard SQL statement groups

- **Given below are the standard SQL statement groups:**

Groups	Statements	Description
DQL	SELECT	DATA QUERY LANGUAGE – It is used to get data from the database and impose ordering upon it.
DML	DELETE INSERT UPDATE MERGE	DATA MANIPULATION LANGUAGE – It is used to change database data.
DDL	DROP TRUNCATE CREATE ALTER	DATA DEFINITION LANGUAGE – It is used to manipulate database structures and definitions.
TCL	COMMIT ROLLBACK SAVEPOINT	TCL statements are used to manage the transactions.
DCL (Rights)	REVOKE GRANT	They are used to remove and provide access rights to database objects.

DBMS/SQL

Data Query Language (The Select Statement)

The Select Statement and Syntax

- **The SELECT command is used to retrieve rows from a single table or multiple Tables or Views.**
  - A query may retrieve information from specified columns or from all of the columns in the Table.
  - It helps to select the required data from the table.

```
SELECT [ALL | DISTINCT] { * | col_name,...}
FROM table_name alias,...
      [ WHERE expr1 ]
      [ CONNECT BY expr2 [ START WITH expr3 ] ]
      [ GROUP BY expr4 ] [ HAVING expr5 ]
      [ UNION | INTERSECT | MINUS SELECT ... ]
      [ ORDER BY expr | ASC | DESC ];
```

Selecting Columns

- **Displays all the columns from the student_master table**

```
SELECT  *
        FROM student_master;
```

- **Displays selected columns from the student_master table**

```
SELECT student_code, student_name
        FROM student_master;
```

The WHERE clause

- **The WHERE clause is used to specify the criteria for selection.**
  - For example: displays the selected columns from the student_master table based on the condition being satisfied

```
SELECT  student_code,  student_name, student_dob
            FROM student_master
            WHERE dept_code = 10;
```

The AS clause

- **The AS clause is used to specify an alternate colum heading.**
  - For example: displays the selected columns from the student_master table based on the condition being satisfied. Observe the column heading

```
SELECT student_dob as "Date of Birth"
            FROM student_master
            WHERE dept_code = 10;

 -- quotes are required when the column heading contains a space
```

```
SELECT student_dob   "Date of Birth"
             FROM student_master
            WHERE dept_code = 10;

-- AS keyword is optional
```

Character Strings and Dates

- **Are enclosed in single quotation marks**
- **Character values are case sensitive**
- **Date values are format sensitive**

```
SELECT  student_code, student_dob
          FROM  student_master
          WHERE student_name = 'Sunil' ;
```

Mathematical, Comparison & Logical Operators

- **Mathematical Operators:**
  - Examples: +, -, *, /
- **Comparison Operators:**

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or Equal to
<	Less than
<=	Less than or Equal to
<>, !=, or ^=	Not Equal to

- **Logical Operators:**
  - Examples: AND, OR, NOT

Other Comparison Operators

Other Comparison operators	Description
[NOT] BETWEEN x AND y	Allows user to express a range. For example: Searching for numbers BETWEEN 5 and 10. The optional NOT would be used when searching for numbers that are NOT BETWEEN 5 AND 10.
[NOT] IN(x,y,…)	Is similar to the OR logical operator. Can search for records which meet at least one condition contained within the parentheses. For example: Pubid IN (1, 4, 5), only books with a publisher id of 1, 4, or 5 will be returned. The optional NOT keyword instructs Oracle to return books not published by Publisher 1, 4, or 5.

Other Comparison Operators

Other Comparison operators	Description
[NOT] LIKE	Can be used when searching for patterns if you are not certain how something is spelt. For example: title LIKE 'TH%'. Using the optional NOT indicates that records that do contain the specified pattern should not be included in the results.
IS[NOT]NULL	Allows user to search for records which do not have an entry in the specified field. For example: Shipdate IS NULL. If you include the optional NOT, it would find the records that do not have an entry in the field. For example: Shipdate IS NOT NULL.

BETWEEN … AND Operator

- **The BETWEEN … AND operator finds values in a specified range:**

```
SELECT staff_code,staff_name
        FROM  staff_master
        WHERE  staff_dob  BETWEEN '01-Jan-1980'
        AND  '31-Jan-1980';
```

IN Operator

- **The IN operator matches a value in a specified list.**
  - The List must be in parentheses.
  - The Values must be separated by commas.

```
SELECT dept_code
      FROM  department_master
      WHERE  dept_name IN ( 'Computer Science', 'Mechanics');
```

LIKE Operator

- **The LIKE operator performs pattern searches.**
  - The LIKE operator is used with wildcard characters.
  - Underscore (_) for exactly one character in the indicated position
  - Percent sign (%) to represent any number of characters

```
SELECT book_code,book_name
        FROM  book_master
        WHERE  book_pub_author LIKE '%Kanetkar%' ;
```

||Operator (Concatenation)

- **The || operator performs concatenation.**
  - between a string literal and a column name.
  - between two column names
  - between string literal and a pseudocolumn

```
SELECT 'Hello' ||  student_name
       FROM  student_master

-- only single quotes not double
```

```
SELECT student_code || '   ' ||  student_name
       FROM  student_master
```

```
SELECT 'Today is ' ||  sysdate
       FROM  dual
```

Logical Operators

- **Logical operators are used to combine conditions.**
  - Logical operators are NOT, AND, OR.
    - NOT reverses meaning.
    - AND both conditions must be true.
    - OR at least one condition must be true.
  - Use of AND operator

```
SELECT staff_code,staff_name,staff_sal
      FROM  staff_master
      WHERE  dept_code = 10
      AND  staff_dob > '01-Jan-1945';
```

Using AND or OR Clause

- **Use of OR operator:**

```
SELECT  book_code
     FROM  book_master
     WHERE  book_pub_author LIKE '%Kanetkar%'
     OR  book  name LIKE '%Pointers%';
```

Using NOT Clause

- **The NOT operator finds rows that do not satisfy a condition.**
  - For example: List staff members working in depts other than 10 & 20.

```
SELECT staff_code,staff_name
      FROM  staff_master
      WHERE  dept_code NOT IN ( 10,20 );
```

Treatment of NULL Values

- **NULL is the absence of data.**
- **Treatment of this scenario requires use of IS NULL operator.**

```
SQL>SELECT student_code
                FROM student_master
                WHERE dept_code IS NULL;
```

Operator Precedence

- **Operator precedence is decided in the following order:**

Levels	Operators
1	* (Multiply), / (Division), % (Modulo)
2	+ (Positive), - (Negative), + (Add), (+ Concatenate), - (Subtract), & (Bitwise AND)
3	=, >, <, >=, <=, <>, !=, !>, !< (Comparison operators)
4	NOT
5	OR
6	AND
7	ALL, ANY, BETWEEN, IN, LIKE, OR, SOME
8	= (Assignment)

The DISTINCT clause

- **The SQL DISTINCT clause is used to eliminate duplicate rows.**
  - For example: Displays student codes from student_marks tables. the student codes are displayed without duplication

```
SELECT DISTINCT student_code
      FROM student_marks;
```

The ORDER BY clause

- **The ORDER BY clause presents data in a sorted order.**
  - It uses an "ascending order" by default.
  - You can use the DESC keyword to change the default sort order.
  - It can process a maximum of 255 columns.
- **In an ascending order, the values will be listed in the following sequence:**
  - Numeric values
  - Character values
  - NULL values
- **In a descending order, the sequence is reversed.**

Sorting Data

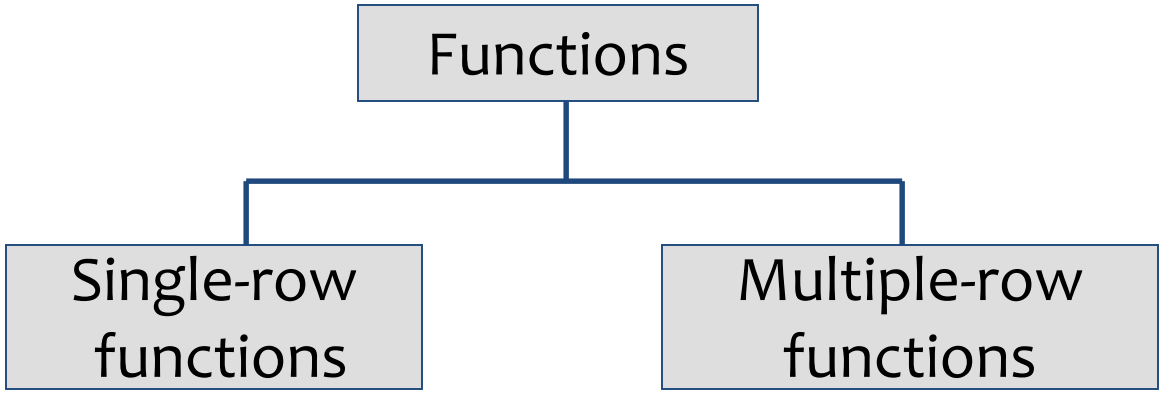
- **The output of the SELECT statement can be sorted using ORDER BY clause**
  - ASC : Ascending order, default
  - DESC : Descending order
- **Display student details from student_master table sorted on student_code in descending order.**

```
SELECT Student_Code,Student_Name,Dept_Code, Student_dob
      FROM Student_Master
      ORDER BY Student_Code DESC ;
```

DBMS SQL

Aggregate (GROUP) Functions

Types of SQL Functions



- **Single row functions :**
  - Operate on single rows only and return one result per row
- **Multiple row functions:**
  - Manipulates groups of rows to give one result per group of rows. Also called as group functions

The Group Functions

- **The Group functions are built-in SQL functions that operate on "groups of rows", and return one value for the entire group.**
- **The results are also based on groups of rows.**
- **For Example, Group function called "SUM" will help you find the total marks, even if the database stores only individual subject marks.**

Syntax : GROUP BY & HAVING clause

- **Syntax**

```
SELECT      [column, ] aggregate function(column), ……
FROM        table
[WHERE      condition]
[GROUP BY column]
[HAVING     condition]
[ORDER BY column] ;
```

Listing of Group Functions

- **Given below is a list of Group functions supported by SQL:**

Function	Value returned
SUM (expr)	Sum value of expr, ignoring NULL values.
AVG (expr)	Average value of expr, ignoring NULL values.
COUNT (expr)	Number of rows where expr evaluates to something other than NULL. COUNT(*) counts all selected rows, including duplicates and rows with NULLs.
MIN (expr)	Minimum value of expr.
MAX (expr)	Maximum value of expr.

Examples of using Group Functions

Example 1: Display the total number of records from student_marks.

```
SELECT COUNT( * )
        FROM Student_Marks;
```

Example 2: Display average marks from each subject.

```
SELECT  AVG(Student_sub1), AVG(Student_sub2), AVG(Student_sub3)
        FROM Student_Marks;
```

The HAVING clause

- **HAVING clause is used to filter data based on the Group functions.**
  - HAVING clause is similar to WHERE condition. However, it is used with Group functions.
- **Group functions cannot be used in WHERE clause. However, they can be used in HAVING clause.**

DBMS SQL

SQL (Single-row) Functions

Single Row Functions - Characteristics

- **Manipulate data items**
- **Accept arguments and return one value**
- **Act on each row returned**
- **Return one result per row**
- **May modify the data type**
- **Can be nested**
- **Accept arguments which can be a column or an expression function_name [(arg1, arg2, …)]**
- **Can be used in SELECT, WHERE, and ORDER BY clauses**

Number Functions

- **Number functions accept "numeric data" as argument, and returns "numeric values".**

TRUNC(arg,n)	Returns a number "arg" truncated to a "n" number of decimal places.
ROUND (arg,n)	Returns "arg" rounded to "n" decimal places. If "n" is omitted, then "arg" is rounded as an integer.
CEIL (arg)	Returns the smallest integer greater than or equal to "arg".
FLOOR (arg)	Returns the largest integer less than or equal to "arg".
ABS (arg)	Returns the absolute value of "arg".
POWER (arg, n)	Returns the argument "arg" raised to the n^{th} power.
SQRT (arg)	Returns the square root of "arg".
SIGN (arg)	Returns –1, 0, or +1 according to "arg" which is negative, zero, or positive respectively.
ABS (arg)	Returns the absolute value of "arg".
MOD (arg1, arg2)	Returns the remainder obtained by dividing "arg1" by "arg2".

Number Functions - Examples

```
SELECT ABS(-15) "Absolute"
    FROM dual;
```

```
SELECT POWER(3,2) "Raised"
    FROM dual;
```

Character Functions

- **Character functions accept "character data" as argument, and returns "character" or "number" values.**

LOWER (arg)	Converts alphabetic character values to lowercase.
UPPER (arg)	Converts alphabetic character values to uppercase.
INITCAP (arg)	Capitalizes first letter of each word in the argument string.
CONCAT (arg1, arg2)	Concatenates the character strings "arg1" and "arg2".
SUBSTR (arg, pos, n)	Extracts a substring from "arg", "n" characters long, and starting at position "pos".
LTRIM (arg)	Removes any leading blanks in the string "arg".
RTRIM (arg)	Removes any trailing blanks in the string "arg".
LENGTH (arg)	Returns the number of characters in the string "arg".
REPLACE (arg, str1, str2)	Returns the string "arg" with all occurrences of the string "str1" replaced by "str2".
LPAD (arg, n, ch)	Pads the string "arg" on the left with the character "ch", to a total width of "n" characters.
RPAD (arg, n, ch)	Pads the string "arg" on the right with the character "ch", to a total width of "n" characters.

Character Functions - Examples

Example 1:

```
SELECT CONCAT('Hello ','World') "Concat"
    FROM Dual;
```

```
Concat
Hello World
```

Example 2:

```
SELECT SUBSTR('HelloWorld',1,5) "SubString"
    FROM Dual;
```

```
SubSt
Hello
```

Date Functions

- **Date Functions operate on Date & Time datatype values**

Add_Months(date1,int1)	Returns a DATE, int1 times added, int1 can be a negative integer
Months_Between(date1,date2	Returns number of months between two dates
Last_Day(date1)	Returns the date of the last day of the month that contains the date
Next_Day(date1,char)	Returns the date of the first weekday specified as char that is later the given date
Current_Date()	Returns the current date in the session time zone. The value is in Gregorian Calendar type
Current_Timestamp	Returns the current date and time in the session time zone. The value returned is of TimeStamp with TimeZone.
Extract(datetime)	Extracts and returns the value of a specified datetime field
Round(date,[fmt])	Returns date rounded to the unit specified . The format will be according to format model fmt
Trunc(date,[fmt])	Returns date truncated to the unit specified . The format will be according to format model fmt

- **Sysdate function returns the current date and time**

Date Functions- Examples

Example 1: To display today's date:

```
SELECT sysdate
    FROM dual;
```

Example 2: To add months to a date:

```
SELECT ADD_MONTHS(sysdate,10)
    FROM dual ;
```

Example 3: To find difference in two dates

```
SELECT MONTHS_BETWEEN(sysdate,'01-sep-95')
    FROM dual ;
```

Arithmetic with Dates

- **Use '+' operator to Add and '-' operator Subtract number of days to/from a date for a resultant date value**

```
SELECT Student_code , (Book_actual_return_date –
    Book_expected_return_date) AS Payable_Days
      FROM  Book_Transaction
       WHERE  Book_Code = 1000;
```

Conversion Functions

- **Conversion functions facilitate the conversion of values from one datatype to another.**

TO_CHAR (arg,fmt)	Converts a number or date “arg” to a specific character format.
TO_DATE (arg,fmt)	Converts a date value stored as string to date datatype
TO_NUMBER (arg)	Converts a number stored as a character to number datatype.
TO_TIMESTAMP(arg,fmt)	Converts character type to a value of timestamp datatype

Conversion Functions - Examples

Example 1: To display system date in format as 29 November, 1999.

```
SELECT TO_CHAR(SYSDATE,'DD month, YYYY') FROM dual ;
```

Example 2: To display system date in the format as 29th November, 1999.

```
SELECT TO_CHAR (SYSDATE,'DDth month,YYYY') FROM dual ;
```

Example 3: To display a number in currency format.

```
select to_char(17000,'$99,999.00')
        FROM dual;
```

Miscellaneous Functions

- **Some functions do not fall under any specific category and hence listed as miscellaneous functions**

NVL (arg1,arg2)	Replaces and returns a null value with specified actual value
NVL2(arg1,arg2,arg3	If arg1 is not null then it returns arg2. If arg1 is null then arg3 is returned
NULLIF(arg1,arg2)	Compares both the arguments, returns null if both are equal or first argument if both are unequal
COALESCE(arg1,arg2 … argn)	Returns the first non null value in the given list
CASE	Both these functions are for conditional processing, with this IF-Then-Else logic can be applied in SQL statements
DECODE	

Miscellaneous Functions - Examples

Example 1: To display the return date of books and if not returned it should display today's date

```
SELECT book_code,
NVL(book_actual_return_date,sysdate)
      FROM book_transaction;
```

Example 2: To examine expected return date of book, and if null return today's date else return the actual return date

```
SELECT book_code,
NVL2(book_expected_return_date,book_actual_return_date, sysdate)
      FROM book_transaction;
```

DBMS/SQL

Joins and Subqueries

What are Joins?

- **If we require data from more than one table in the database, then a join is used.**
  - Tables are joined on columns, which have the same “data type” and “data width” in the tables.
  - The JOIN operator specifies how to relate tables in the query.
    - When you join two tables a Cartesian product is formed, by default.
  - Oracle supports
    - Oracle Proprietary
    - SQL: 1999 Compliant Joins

Types of Joins

- **Given below is a list of JOINs supported by Oracle:**

Oracle Proprietary Joins	SQL: 1999 Compliant Joins
Cartesian Product	Cross Joins
Equijoin	Inner Joins (Natural Joins)
Outer-join	Left, Right, Full outer joins
Non-equijoin	Join on
Self-join	Join on

Cartesian Joins

- **A Cartesian product is a product of all the rows of all the tables in the query.**
- **A Cartesian product is formed when the join condition is omitted or it is invalid**
- **To avoid having Cartesian product always include a valid join condition**

Example

```
SELECT Student_Name, Dept_Name
        FROM  Student_Master, Department_Master;
```

Guidelines for Joining Tables

- **The JOIN condition is written in the WHERE clause**
- **The column names which appear in more than one table should be prefixed with the table name**
- **To improve performance of the query, table name prefix can be include for the other selected columns too**

EquiJoin

- **In an Equijoin, the WHERE statement compares two columns from two tables with the equivalence operator "=".**
- **This JOIN returns all rows from both tables, where there is a match.**
- **Syntax :**

```
SELECT <col1>, <col2>,…
        FROM <table1>,<table2>
        Where <table1>.<col1>=<table2>.<col2>
        [AND <condition>] [ORDER BY <col1>, <col2>,… ]
```

EquiJoin - Example

Example 1: To display student code and name along with the department name to which they belong

```
SELECT Student_Code,Student_name,Dept_name
  FROM Student_Master ,Department_Master
  WHERE Student_Master.Dept_code =Department_Master.Dept_code;
```

Example 2: To display student and staff name along with the department name to which they belong

```
SELECT student_name,staff_name, dept_name
        FROM student_master, department_master,staff_master
        WHERE student_master.dept_code=department_master.dept_code
         and staff_master.dept_code=department_master.dept_code;
```

Outer Join

- **If a row does not satisfy a JOIN condition, then the row will not appear in the query result.**
- **The missing row(s) can be returned by using OUTER JOIN operator in the JOIN condition.**
- **The operator is PLUS sign enclosed in parentheses (+), and is placed on the side of the join(table), which is deficient in information.**

Outer Join

- **Syntax**
- **Table1.column = table2.column (+) means OUTER join is taken on table1.**
- **The (+) sign must be kept on the side of the join that is deficient in information**
- **Depending on the position of the outer join (+), it can be denoted as Left Outer or Right outer Join**

WHERE table1 <OUTER JOIN INDICATOR> = table 2

Outer Join - Example

- **To display Department details which have staff members and also display department details who do not have any staff members**

```
SELECT staff.staff_code,staff.Dept_Code,dept.Dept_name
    FROM Staff_master staff, Department_Master dept
    WHERE staff.Dept_Code(+) = dept.Dept_Code
```

LEFT, RIGHT & FULL Outer Join

- **A join between two tables that return rows that match the join condition and also unmatched rows from left table is LEFT OUTER JOIN**
- **A join between two tables that return rows that match the join condition and unmatched rows from the right table is RIGHT OUTER JOIN**
- **A join between two tables that return rows that match the join condition and returns unmatched rows of both left and right table is a full outer join**

LEFT, RIGHT & FULL Outer Join - Example

- **Example 1: Display student & department details and also thos departments who do have students**

```
SELECT s.student_code, s.dept_code, d.dept_name
        FROM student_master s
        RIGHT OUTER JOIN department_master d
        ON (s.dept_code = d.dept_code);
```

- **Example 2 Display student & department details, also those students who are not assigned to any department**

```
SELECT s.student_code, s.dept_code, d.dept_name
        FROM student_master s
        LEFT OUTER JOIN department_master d
        ON (s.dept_code = d.dept_code);
```

LEFT, RIGHT & FULL Outer Join - Example

- **Example 3: Display student & department details. Also those departments who do have students and students who are not assigned to any department**

```
SELECT s.student_code,s.dept_code,d.dept_name
        FROM student_master s
        FULL OUTER JOIN department_master d
        ON (s.dept_code = d.dept_code );
```

What is a SubQuery?

- **A sub-query is a form of an SQL statement that appears inside another SQL statement.**
  - It is also called as a “nested query”.
- **The statement, which contains the sub-query, is called the “parent statement”.**
- **The “parent statement” uses the rows returned by the sub-query.**

Subquery - Examples

Example 1: To display name of students from "Mechanics" department.

- Method 1:

```
SELECT Dept_Code
    FROM Department_Master
    WHERE Dept_name = 'Mechanics';
```

O/P : 40

```
SELECT student_code,student_name
FROM student_master
WHERE dept_code=40;
```

Subquery - Examples

Example 1 (contd.):

- Method 2: Using sub-query

```
SELECT student_code, student_name
        FROM student_master
       WHERE dept_code = (SELECT dept_code
                          FROM department_master
                         WHERE dept_name = 'Mechanics');
```

Where to use Subqueries?

- **Subqueries can be used for the following purpose :**
  - To insert records in a target table.
  - To create tables and insert records in the table created.
  - To update records in the target table.
  - To create views.
  - To provide values for conditions in the clauses, like WHERE, HAVING, IN, etc., which are used with SELECT, UPDATE and DELETE statements.

Comparison Operators for Subqueries

- **Types of SubQueries**
  - Single Row Subquery
  - Multiple Row Subquery.
- **Some comparison operators for subqueries:**

Operator	Description
IN	Equals to any member of
NOT IN	Not equal to any member of
*ANY	compare value to every value returned by sub-query using operator *
*ALL	compare value to all values returned by sub-query using operator *

Using Comparison Operators - Examples

- **Example 1: To display all staff details of who earn salary least salary**

```
SELECT staff_name, staff_code, staff_sal
        FROM staff_master
        WHERE staff_sal  = (SELECT MIN(staff_sal)
                                    FROM staff_master) ;
```

- **Example 2: To display staff details who earn salary greater than average salary earned in dept 10**

```
SELECT staff_code,staff_sal
          FROM staff_master
          WHERE staff_sal > ANY(SELECT AVG(staff_sal)
                                      FROM staff_master WHERE dept_code=10);
```

What is a Co-related Subquery?

- **A sub-query becomes "co-related", when the sub-query references a column from a table in the "parent query".**
  - A co-related sub-query is evaluated once for each row processed by the "parent statement", which can be either SELECT, UPDATE, or DELETE statement.
  - A co-related sub-query is used whenever a sub-query must return a "different result" for each "candidate row" considered by the "parent query".

Co-related Subquery -Examples

Example 2: To display staff details whose salary is greater than the average salary in their own department:

```
SELECT staff_name, staff_sal , dept_code
   FROM staff_Master  s
   WHERE  staff_sal > (SELECT AVG(staff_sal)
                       FROM staff_Master m
                          WHERE  s.dept_code  = m.dept_code );
```

EXISTS/ NOT EXISTS Operator

- **The EXISTS / NOT EXISTS operator enables to test whether a value retrieved by the Outer query exists in the result-set of the values retrieved by the Inner query.**
  - The EXISTS / NOT EXISTS operator is usually used with a co-related sub-query.
    - If the query returns at least one row, the operator returns TRUE.
    - If the value does not exist, it returns FALSE.
  - The NOT EXISTS operator enables to test whether a value retrieved by the Outer query is not a part of the result-set of the values retrieved by the Inner query.

EXISTS/ NOT EXISTS Operator - Examples

- **Example 1: To display details of employees who have some other**

```
SELECT staff_code, staff_name
    FROM staff_master staff
    WHERE  EXISTS (SELECT mgr_code FROM staff_master mgr WHERE
    mgr.mgr_code = staff.staff_code) ;
```

- **Example 2: To display details of departments which have**

```
SELECT dept_code,dept_name
    FROM department_master
    WHERE  EXISTS ( SELECT dept_code FROM staff_master
                    WHERE staff_master.dept_code =
                    department_master.dept_code) ;
```

DBMS/SQL

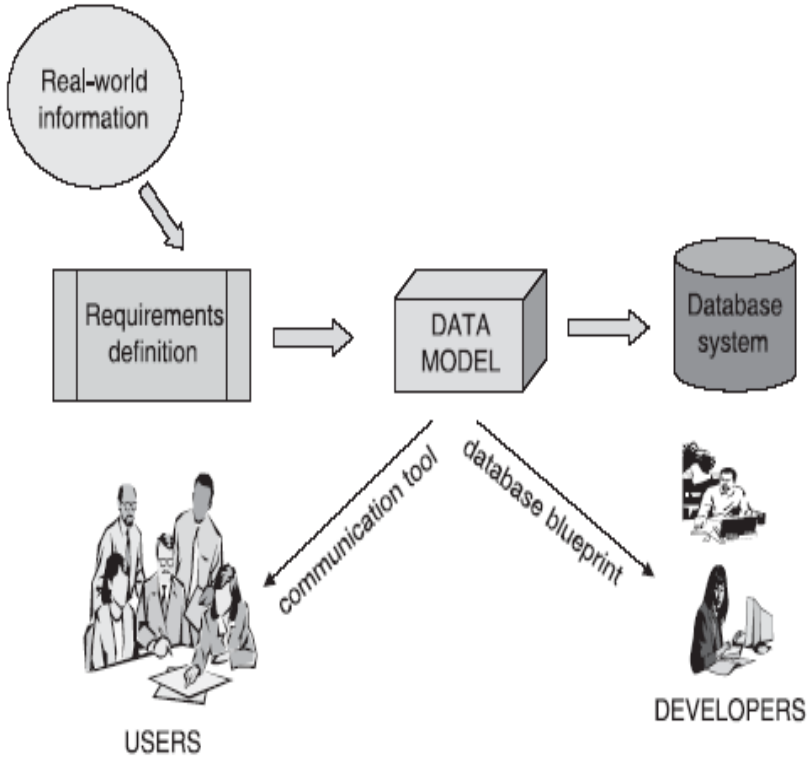
Introduction to Data Modeling , E-R Model and Normalization

Definition of a Model

- **An integrated collection of concepts for describing data, relationships between data, and constraints on the data used by an organization.**
- **A representation of 'real world' objects and events, and their associations.**
- **It attempts to represent the data requirements of the organization that you wish to model.**
- **Modeling is an integral part of the design and development of any system.**
- **A correct model is essential.**

What is Data Modeling?

- **Data modeling is a technique for exploring the data structures needed to support an organization's information need.**
- **It would be a conceptual representation or a replica of the data structure required in the database system.**
- **A data model focuses on which data is required and how the data should be organized.**
- **At the conceptual level, the data model is independent of any hardware or software constraints.**



Why Use Data Modeling?

- **Leverage**
  - Data model serves as a blueprint for the database system.
- **Conciseness**
  - Data model functions as an effective communication tool for discussions with the users.
- **Data Quality**
  - Data model acts as a bridge from real-world information to database storing relevant data content.

Entity-Relationship Model

- **A high-level conceptual data model that is widely used in the design of a database application.**
- **A top-down approach to database design.**
- **The ER model represents data in terms of these:**
  - Entities (Independent, dependent or Associative)
  - Attributes of entities
  - Relationships between entities
- **It is independent of the h/w or s/w used for implementation. Although we can use an ER model as a basis for hierarchical, network and relational databases, it is strongly connected to the relational databases.**

Entities

- **People, places, objects, things, events, or concepts (nouns).**
  - Examples: Employee, salesperson, sale, account, department.
- **Represented as a rectangle in ERD.**
- **Entity type versus entity instance:**
  - Entity instance: A single example of an entity.
  - Entity type: A collection of all entity instances of a single type.

Attributes

- **Attributes describe the entity with which they are associated.**
- **They are characteristics of an Entity.**

- **Eg: Customer Name is an attribute of the entity Customer.**

Types of Attributes

- **Simple Attribute: An attribute composed of a single component with an independent existence. E.g SSN, BirthDate.**

- **Composite Attribute: An attribute composed of multiple components, each with an independent existence. E.g Name attribute of the entity that can be subdivided into FirstName ,LastName attributes.**

Types of Attributes

- **Single-Valued Attribute: An attribute that holds a single value for each occurrence. E.g.SSN**
- **Multi-Valued Attributes: An attribute that holds multiple values for each occurrence. E.g Hobbies**
- **Derived Attributes: An attribute that represents a value that is derivable from the value of a related attribute or set of attributes, not necessarily in the same entity type.**
- **E.g Age whose value is derived from the Current Date and BirthDate attribute.**

Relationship

- **A relationship represents an association between two or more entities. An example of a relationship would be as follows:**
  - Employees are assigned to projects.
  - Projects have subtasks.
  - Departments manage one or more projects.

Key Types

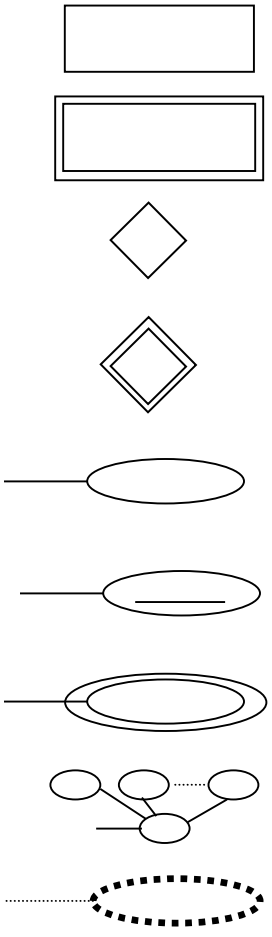
- **Superkey:**
- **An attribute, or set of attributes, that uniquely identifies each entity occurrence.**
- **If we add additional attributes to a primary key, the resulting combination would still uniquely identify an instance of the entity set. Such augmented keys are called superkey.**

- **Candidate Key:**
- **A superkey that contains only the minimum number of attributes necessary for unique identification of each entity occurrence. Ex. BranchNo in entity Branch.**
- **The minimal super key is also referred as candidate key.**

Key Types

- **Primary Key: The candidate key that is selected to uniquely identify each occurrence of an entity type.E.g: National Insurance Number.**
- **Alternate keys: The candidate keys that are not selected as the primary key of the entity.**
- **Composite Key: A candidate key that consist of two or more attributes.**

Notations Used for ER Model



ENTITY TYPE

WEAK ENTITY TYPE

RELATIONSHIP TYPE

IDENTIFYING RELATIONSHIP TYPE

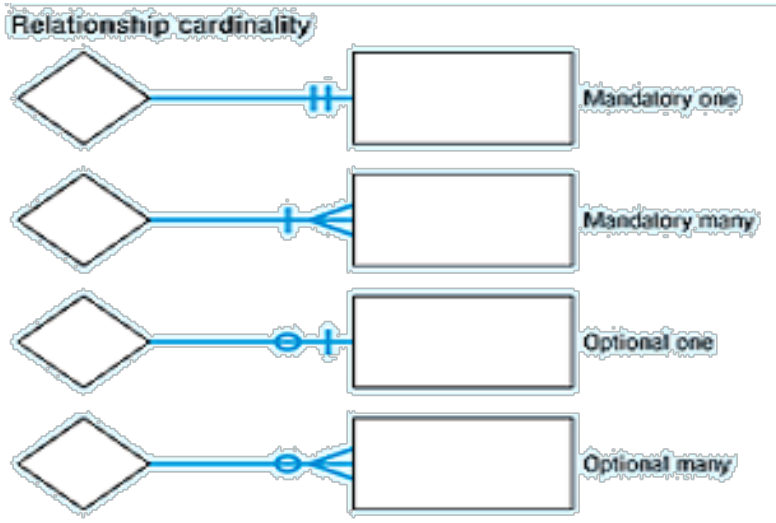
ATTRIBUTE

KEY ATTRIBUTE

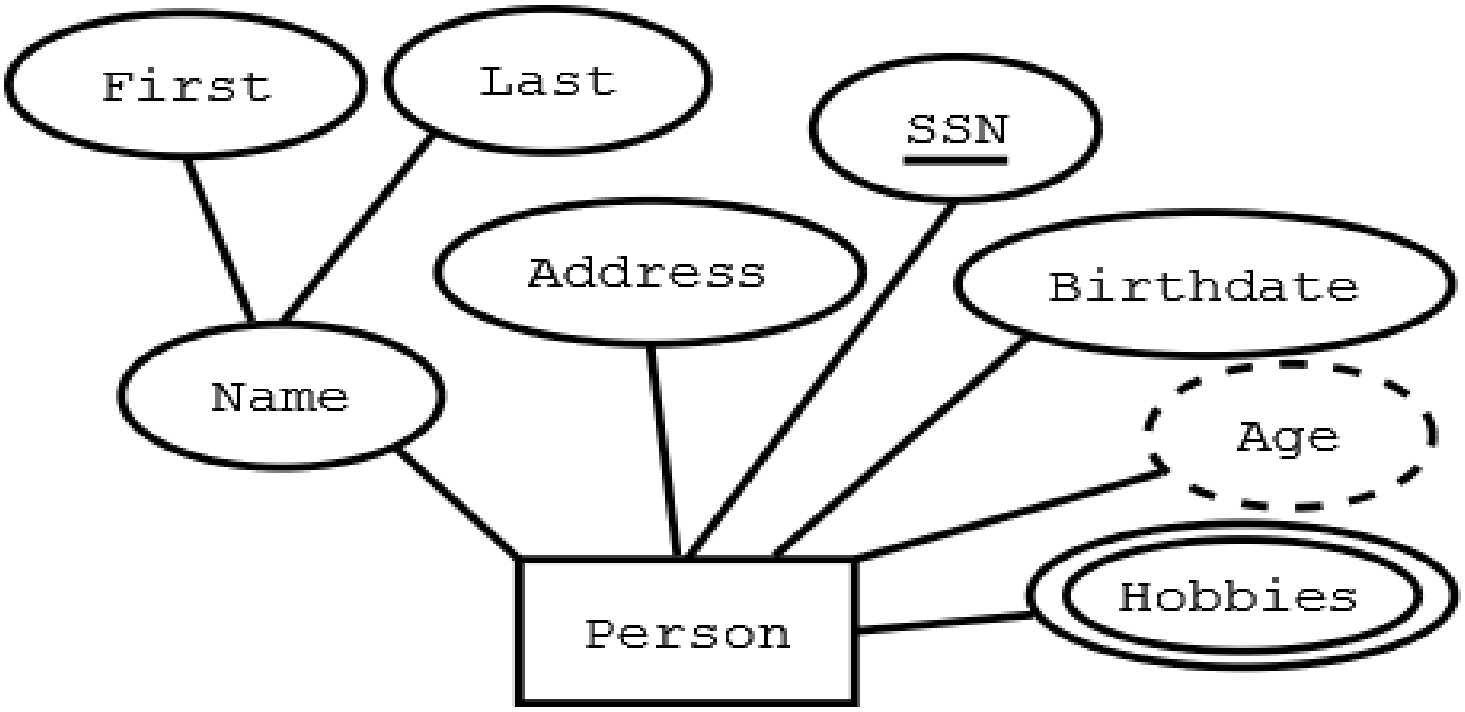
MULTIVALUED ATTRIBUTE

COMPOSITE ATTRIBUTE

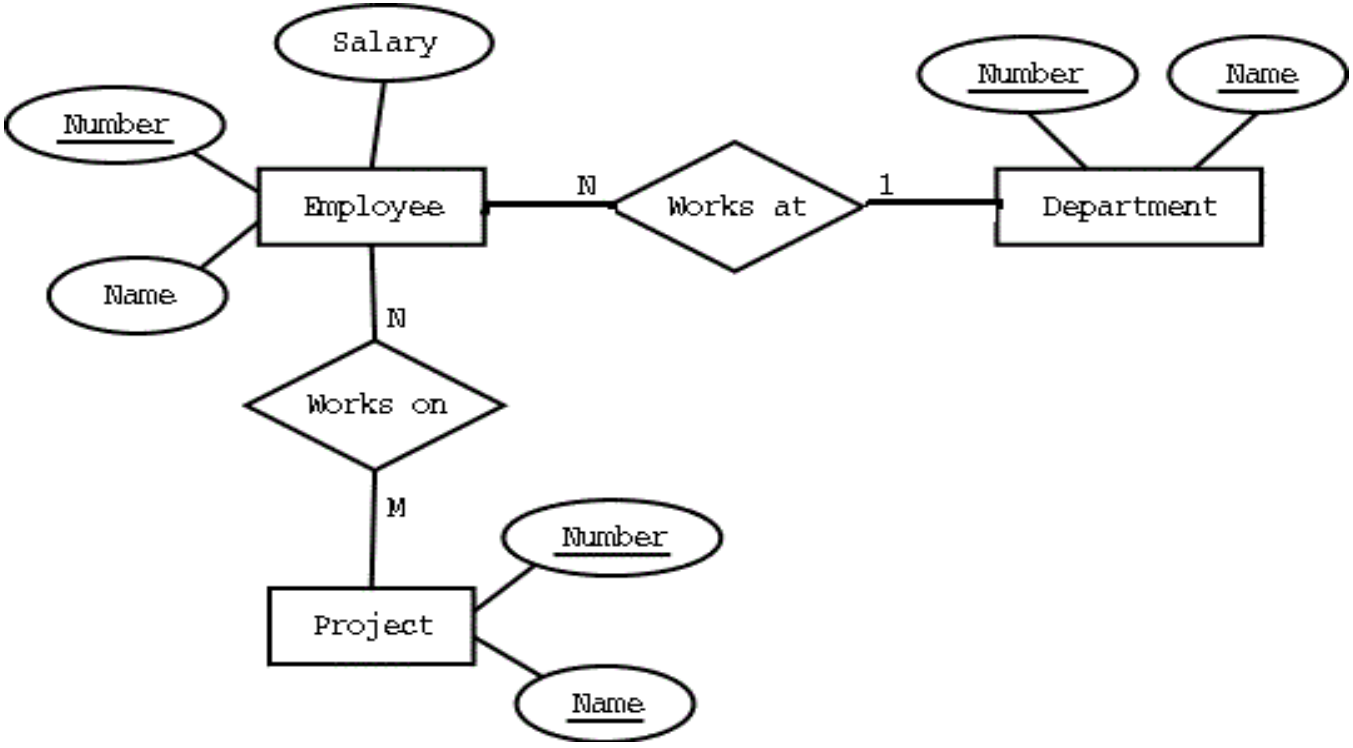
DERIVED ATTRIBUTE



Attribute Notations



The E-R Model - An Example



The Different ER Relationships (cont..)

- **Degree of Relationships**
  - The degree of a relationship is the number of entities associated with the relationship.

Types of Binary Relationships

- **One-to-one (1:1) binary relationship:**
  - One instance of an entity type is related to only one instance of another entity type, and vice versa.
    - Line 3.1.1
  - Example: A salesperson can have only ONE office; an office can be assigned to only ONE salesperson.
- **One-to-many (1:N) binary relationship:**
  - One instance of an entity type (the "one" side) is related to many instances of another entity type (the "many" side), but an entity instance on the "many" side can be related to only one instance on the "one" side.
  - Example: A salesperson can have MANY sales transaction, but one sales transaction can only be done by ONE salesperson.

Types of Binary Relationships

- **Many-to-many (M:N) binary relationship:**
  - Instances of each entity type can be related to many instances of the other entity type.
  - Example: A salesperson can sell MANY products, and a product can be sold by MANY salespersons.

Normalization Concepts

- **Normalization is a technique for designing relational database tables.**
- **Normalization is used:**
  - to minimize duplication of information
  - to safeguard the database against certain types of problems
- **Thus, Normalization is a process of efficiently organizing data in a database.**
- **Normalization is done within the framework of five normal forms(numbered from first to fifth).**
- **Most designs conform to at least the third normal form.**
- **Normalization rules are designed to prevent anomalies and data inconsistencies.**

Goals of the Normalization

- **Goals of the Normalization process are:**
  - to eliminate redundant data
  - to ensure that data dependencies make sense
- **Thus Normalization:**
  - reduces the amount of space a database has consumed
  - ensures that data is logically stored

Benefits of Normalization

- **Benefits of Normalization are given below:**
  - Greater overall database organization
  - Reduction of redundant data
  - Data consistency within the database
  - A much more flexible database design
  - A better handle on database security

Problems with un-normalized database

- **An un-normalized database can suffer from various “logical inconsistencies” and “data operation anomalies”.**
- **Data Operation anomalies can be classified as:**
  - Update anomaly
  - Insertion anomaly
  - Deletion anomaly

Update anomaly

- **The slide shows an Update anomaly.**
- **Employee 519 is shown as having different addresses on different records.**

Employees' Skills

Employee ID	Employee Address	Skill
426	87 Sycamore Grove	Typing
426	87 Sycamore Grove	Shorthand
519	94 Chestnut Street	Public Speaking
519	96 Walnut Avenue	Carpentry

Insertion anomaly

- **The slide shows an example of an Insertion anomaly.**
- **Until the new faculty member is assigned to teach at least one course, the details cannot be recorded.**

Faculty and Their Courses

Faculty ID	Faculty Name	Faculty Hire Date	Course Code
389	Dr. Giddens	10-Feb-1985	ENG-206
407	Dr. Saperstein	19-Apr-1999	CMP-101
407	Dr. Saperstein	19-Apr-1999	CMP-201
424	Dr. Newsome	29-Mar-2007	?

Deletion anomaly

- **The slide shows an example of a Deletion anomaly.**
- **All information about Dr. Giddens is lost when he temporarily ceases to be assigned to any course.**

Faculty and Their Courses

Faculty ID	Faculty Name	Faculty Hire Date	Course Code
389	Dr. Giddens	10-Feb-1985	ENG-206
407	Dr. Saperstein	19-Apr-1999	CMP-101
407	Dr. Saperstein	19-Apr-1999	CMP-201

DELETE

Case Study: Normalization

- **Given an EmpProject Table structure**
- **Note: EmpProject is an un-normalized relation**

Proj Code	Proj Type	Proj Desc	Empno	Ename	Grade	Sal scale	Proj Join Date	Alloc Time
001	APP	LNG	46	JONES	A1	5	12/1/1998	24
001	APP	LNG	92	SMITH	A2	4	2/1/1999	24
001	APP	LNG	96	BLACK	B1	9	2/1/1999	18
004	MAI	SHO	72	JACK	A2	4	2/4/1999	6
004	MAI	SHO	92	SMITH	A2	4	5/5/1999	6

Normalization

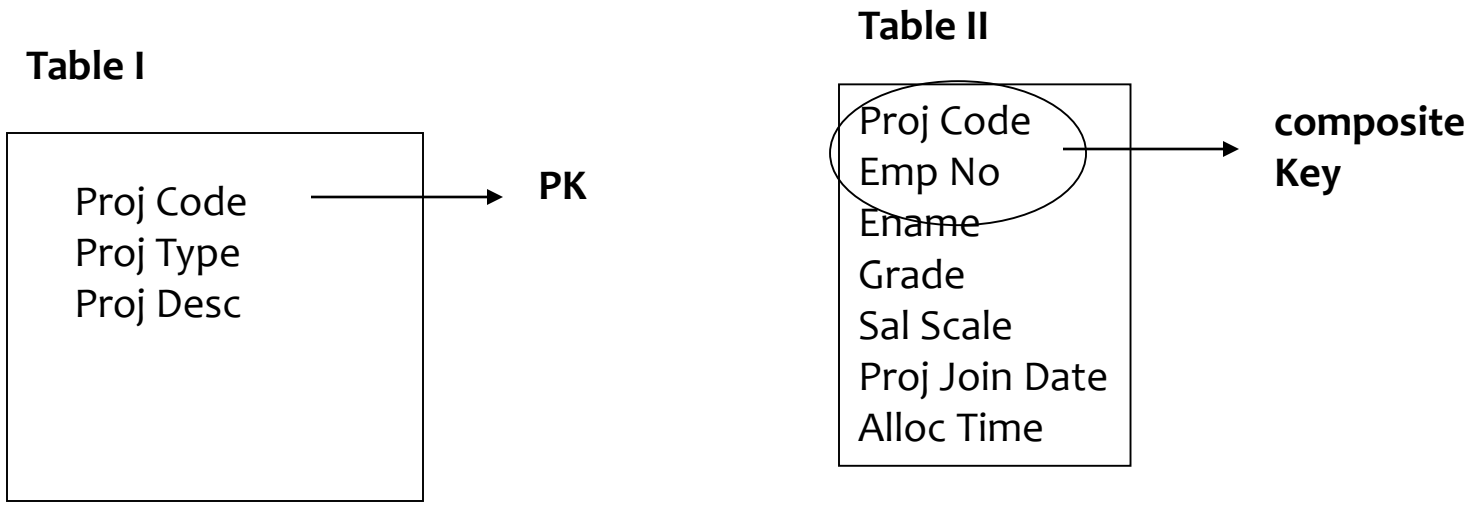
- **Problems with the above kind of implementation**
  - Data Redundancy : Proj. type and description are unnecessarily repeated many times.
  - An employee can't logically exist.
  - If project type changes many records will require updating in an identical manner.
  - If employee Smith gains promotion, many records will require updating.
  - If management decides to change the relationship between the grade and salary scale then many updates will be required.

Normalization – First Normal Form (1 NF)

- **A relation is said to be in "First Normal Form" (1NF) if and only if all its attributes assume only atomic values and there are no repeating groups.**
- **1NF requires that the values in each column of a table are "atomic".**
  - "Atomic" implies that there are no sets of values within a column.
- **To Transform un-normalized relation in 1 NF:**
- **Eliminate repeating groups.**
- **Make a separate table for each set of related attributes, and give each table a primary key.**

Normalization – First Normal Form (1 NF)

- **Rule:**
  - A relation is in the first normal form if the intersection of any column and row contains only one value
  - Identify any suitable primary key
  - Remove repeating groups to simplify relationship that may lead to multiple table design



Normalization – Second Normal Form (2 NF)

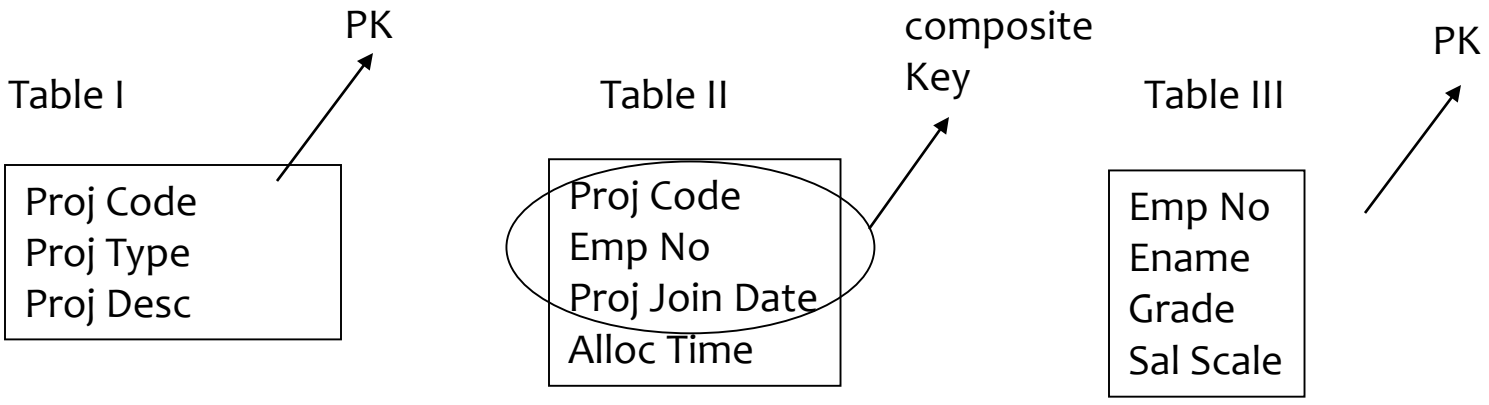
- **A relation is in Second Normal Form (2NF) if and only if it is in 1NF and every non-key attribute is fully functionally dependent on the complete primary key of the relation.**

- **2NF is based on the concept of Full Functional Dependency.**

Normalization – Second Normal Form (2 NF)

- **Transform 1 NF to 2 NF :**
- **If a non-key attribute depends on only part of a composite key, remove it to a separate table.**
  - For every relation with a single data item making up the primary key, this rule should "always be true".
  - For those with the composite key, examine every column and ask whether its value depends on the whole of the composite key or just some part of it.
  - Remove those that depend only on part of the key to a new relation with that part as the primary key.

Normalization – Second Normal Form (2 NF)



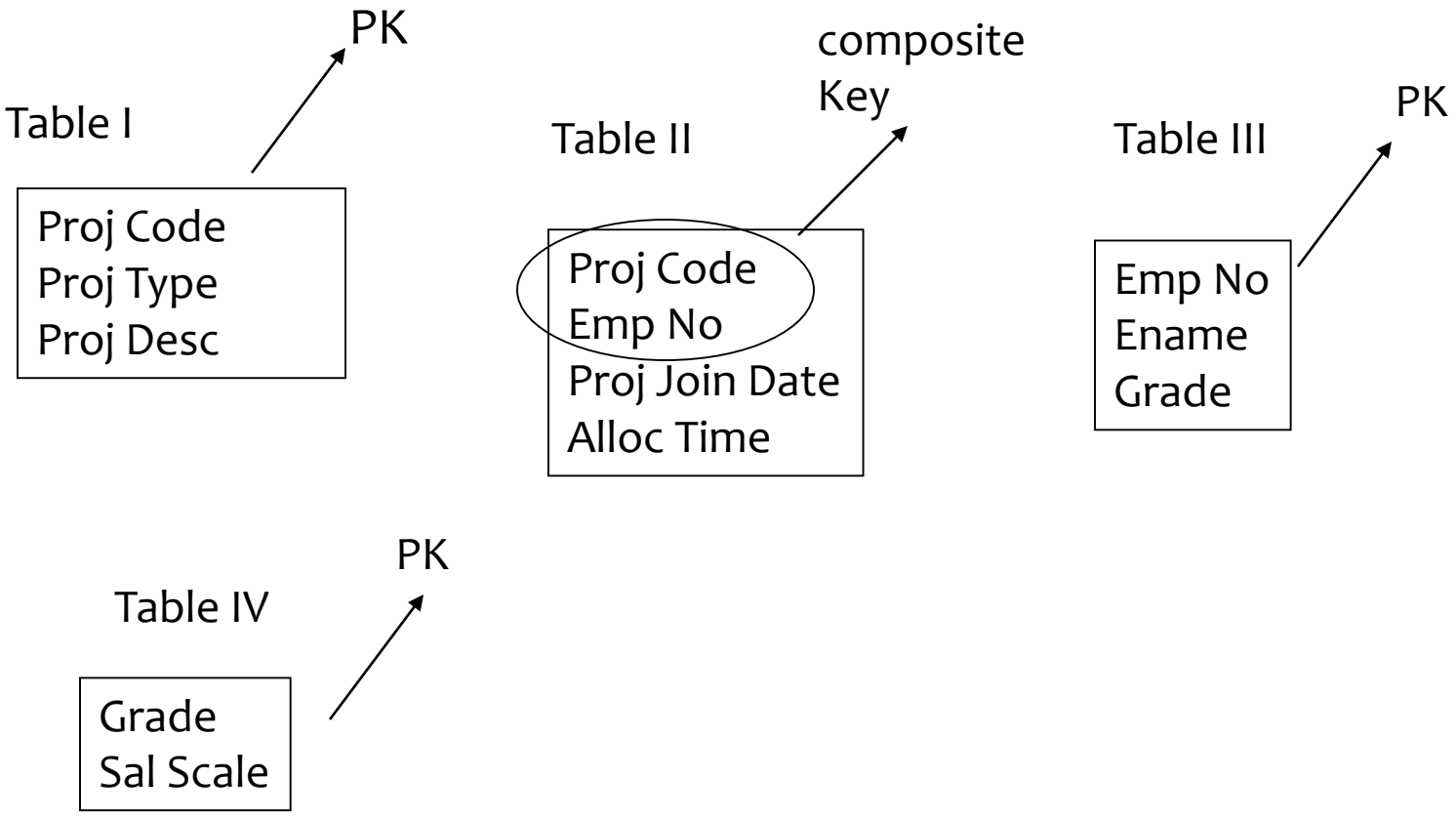
Normalization – Third Normal Form (3 NF)

- **A relation is in 3NF if and only if it is in 2NF and every non-key attribute is non-transitively dependent on the primary key of the relation.**
- **3NF is based on the concept of transitive dependency.**

Normalization – Third Normal Form (3 NF)

- **Rules for 3NF are:**
  - Examine every non-key column and question its relationship with every other non-key column.
  - If there is a transitive dependency, then remove both the columns to a new relation.

Normalization – Third Normal Form (3 NF)



Normalization Summarization

- **1 NF- Ensure all values are atomic and Eliminate Repeating Groups**
- **2 NF- Eliminate Partial Dependencies**
- **3 NF- Eliminate Transitive Dependencies**

Drawbacks of Normalization

- Typically, in a normalized database, more joins are required to pull together information from multiple tables.
- Joins require additional I/O to process, and are therefore more expensive from a performance standpoint than single-table lookups.
- Additionally, a normalized database often incurs additional CPU processing. CPU resources are required to perform join logic and to maintain data and referential integrity.

DBMS/SQL

Database Objects

Lesson Objectives

- **To understand the following Database Objects:**
  - Basic Data Types
  - Data Integrity
  - Examples of CREATE TABLE
  - Examples of ALTER TABLE
  - Database Objects
  - Index
  - Synonym
  - Sequence
  - View
  - Deleting Database Objects

Overview

- **A database is a collection of structures with appropriate authorizations and accesses that are defined.**
- **The structures in the database like tables, indexes, etc. are called as objects in the database.**
- **All objects that belong to the same user are said to be the "schema" for the particular user.**
- **Information about existing objects can be retrieved from dba_/user_/all_objects.**

Basic Data Types

- **Given below are the basic Data Types:**

Datatype	Description
CHAR(n)	Stores fixed length string. Maximum length = 2000 bytes For example: NAME CHAR(15)
VARCHAR2(n)	Stores variable length string. Maximum length = 4000 bytes For example: DESCRIPTION VARCHAR2(100)
LONG(n)	Stores variable length string . Maximum length = 2 GIGA bytes For example: SYNOPSIS LONG(5000)
NUMBER(p,s)	Stores numeric data . Range is 1E-129 to 9.99E125 Max Number of significant digits = 38 For example: SALARY NUMBER(9,2)
DATE	Stores DATE. Range from January 1, 4712 BC to December 31, 9999 AD. Both DATE and TIME are stored. Requires 7 bytes. For example: HIREDATE DATE
RAW(n)	Stores data in binary format such as signature, photograph. Maximum size = 255 bytes
LONG RAW(n)	Same as RAW. Maximum size = 2 Gigabytes

Basic Data Types contd..

Datatype	Description
TIMESTAMP	**Stores the time to be stored as a date with fractional seconds. Extension to the DATA datatype** **There are some variations of the data type**

Basic Data Types contd..

Datatype	Description
BLOB	Binary Large Object • **Stores any kind of data in binary format.** • **Typically used for multimedia data such as images, audio, and video.**
CLOB	Character Large Object • **Stores string data in the database character set format. Used for large strings or documents that exclusively use the database character set.** • **Characters in the database character set are in a fixed width format.**
NCLOB	National Character Set Large Object • **Stores string data in National Character Set format. Used for large strings or documents in the National Character Set.** • **Supports characters of varying width format.**
BFILE	External Binary File • **A binary file stored outside the database in the host operating system file system, but accessible from database tables.** • **BFILEs can be accessed from your application on a read-only basis. Use BFILEs to store static data, such as image data, that does not need to be manipulated in applications.** • **Any kind of data, that is, any operating system file, can be stored in a BFILE. For example: You can store character data in a BFILE, and then load the BFILE data into a CLOB specifying the character set upon loading.**

Table

- **Tables are objects, which store the user data.**
- **Use the CREATE TABLE statement to create a table, which is the basic structure to hold data.**

For example:

```
CREATE TABLE book_master
(book_code number,
book_name varchar2(50),
book_pub_year number,
book_pub_author varchar2(50));
```

Applying Constraints

- **Constraints can be defined at**
  - Column Level

```
CREATE TABLE  tablename
(column datatype  [DEFAULT expr] [column_constraint] ,
……)
```

  - Table Level

```
CREATE TABLE  tablename
(column datatype,
 column datatype
……
 [CONSTRAINT constraint_name] constraint_type
(column,…))
```

Types of Integrity Constraints

- **Let us see the types of Data Integrity Constraints:**
  - Nulls
  - Unique Column Values
  - Primary Key Values
  - Referential Integrity

NOT NULL Constraint

- **The user will not be allowed to enter null value.**

For Example:

- A NULL value is different from a blank or a zero. It is used for a quantity that is "unknown".
- A NULL value can be inserted into a column of any data type.

```
CREATE TABLE  student_master
(student_code  number(4) NOT NULL,
dept_code   number(4) CONSTRAINT dept_code_nn
                        NOT  NULL );
```

DEFAULT clause

- **If no value is given, then instead of using a "Not Null" constraint, it is sometimes useful to specify a default value for an attribute.**

For Example:

- When a record is inserted the default value can be considered.

```
CREATE TABLE  staff_master(
Staff_Code number(8) PRIMARY KEY,
Staff_Name varchar2(50) NOT NULL,
Staff_dob date,
Hiredate date DEFAULT sysdate,
…..)
```

UNIQUE constraint

- **The keyword UNIQUE specifies that no two records can have the same attribute value for this column.**

For Example:

```
CREATE TABLE student_master
(student_code number(4),
 student_name varchar2(30)
CONSTRAINT stu_id_uk  UNIQUE(student_code )) ;
```

PRIMARY KEY constraint

- **The Primary Key constraint enables a unique identification of each record in a table.**

For Example:

```
CREATE TABLE Staff Master
(staff_code  number(6)
CONSTRAINT staff_id_pk PRIMARY KEY
 staff_name varchar2(20)
 ……. );
```

CHECK constraint

- **CHECK constraint allows users to restrict possible attribute values for a column to admissible ones.**

For Example:

```
CREATE TABLE staff_master
( staff_code number(2),
  staff_name  varchar2(20),
  staff_sal   number(10,2) CONSTRAINT staff_sal_min
                                CHECK (staff_sal >1000),
  …..) ;
```

FOREIGN KEY constraint

- **The FOREIGN KEY constraint specifies a "column" or a "list of columns" as a foreign key of the referencing table.**
- **The referencing table is called the "child-table", and the referenced table is called "parent-table".**

For Example:

```
CREATE TABLE student_master
(student_code number(6) ,
 dept_code number(4) CONSTRAINT stu_dept_fk
REFERENCES department_master(dept_code),
 student_name varchar2(30) );
```

Create new table based on existing table

- **Constraints on an “old table” will not be applicable for a “new table”.**

```
CREATE TABLE student_dept117 AS
SELECT student_code, student_name
FROM student_master WHERE  dept_code = 117
```

ALTER Table

- **Given below is an example of ALTER TABLE:**

```
ALTER TABLE table_name
        [ADD (col_name col_datatype col_constraint ,...)]|
        [ADD (table_constraint)]|
        [DROP CONSTRAINT constraint_name]|
        [MODIFY existing_col_name new_col_datatype
                                        new_constraint new_default]
        [DROP COLUMN existing_col_name]
        [SET UNUSED COLUMN existing_col)name];
```

ALTER Table – Add clause

- **The "Add" keyword is used to add a column or constraint to an existing table.**
  - For adding three more columns to the emp table, refer the following example:

```
ALTER TABLE Student_Master
ADD (last_name varchar2(25) );
```

ALTER Table – Add clause

- For adding Referential Integrity on “mgr_code" column, refer the following example:

```
ALTER TABLE staff_master
  ADD CONSTRAINT FK FOREIGN KEY  (mgr_code) REFERENCES
staff_master(staff_code);
```

ALTER Table – MODIFY clause

- **MODIFY clause:**
  - The "Modify" keyword allows making modification to the existing columns of a table.
    - For Modifying the width of "sal" column, refer the following example:

```
ALTER TABLE staff_master
MODIFY (staff_sal number (12,2) ) ;
```

ALTER Table – DROP clause

- **The DROP clause is used to remove constraints from a table.**
  - For Dropping the FOREIGN KEY constraint on “department”, refer the following example:

```
ALTER TABLE  student_master
DROP CONSTRAINT  stu_dept_fk ;
```

Dropping Column

- Directly dropping the columns.

```
ALTER TABLE staff_master DROP COLUMN staff_sal;
```

Drop a Table

- **The DROP TABLE command is used to remove the definition of a table from the database.**

For Example:

```
DROP TABLE staff_master;

DROP TABLE Department_master
    CASCADE CONSTRAINTS;
```

User_Tables & User_Objects

- **To view the names of tables owned by the user, use the following query:**
- **To view distinct object types owned by the user, use the following query:**

```
SELECT table_name
FROM user_tables
```

```
SELECT DISTINCT object_type
FROM user_objects ;
```

Usage of Index

- **Index is a database object that functions as a “performance-tuning” method for allowing faster retrieval of records.**
- **Index creates an entry for each value that appears in the indexed columns.**
- **The absence or presence of an Index does not require change in wording of any SQL statement.**

Usage of Index

- **Syntax:**

```
CREATE [UNIQUE] INDEX index_name
ON table_name(col_name1 [ASC|DESC],col_name2,.....)
```

Creating an Index

Example 1: A simple example of an Index is given below:

```
CREATE INDEX  staff_sal_index ON staff_master(staff_sal);
```

Example 2: To allow only unique values in the field "ename", the CREATE statement should appear as shown below:

```
CREATE UNIQUE INDEX  staff_ename_unindex
ON staff_master(staff_name );
```

How are Indexes created?

- **Indexes can be either created “automatically” or “manually”.**
  - **Automatically:** A unique Index is automatically created when you define a PRIMARY KEY or UNIQUE constraint in a table definition.
  - **Manually:** A non-unique index can be created on columns by users in order to speed up access to the rows.

Usage of View

- **A View can be thought of as a “stored query” or a “virtual table”, i.e. a logical table based on one or more tables.**
  - A View can be used as if it is a table.
  - A View does not contain data.

Usage of View

- **Syntax**

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view [(alias[, alias]...)]
AS subquery
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY [CONSTRAINT constraint]];
```

Creating a View

- **Given below is an example of a simple View:**

```
CREATE VIEW staff_view
        AS
        SELECT * FROM staff_master
        WHERE hiredate >'01-jan-82';
```

Creating a View

- **Creating a Complex View:**
  - As shown in the example given below, create a Complex View that contains group functions to display values from two tables.

```
CREATE VIEW dept_sum_vu
 (name, minsal, maxsal, avgsal)
 AS SELECT dept.dept_name, MIN(staff.staff_sal),
          MAX(staff. staff_sal),AVG(staff. staff_sal)
  FROM   staff_master staff, department_master dept
  WHERE staff.dept_code = dept.dept_code
  GROUP BY dept.dept_name;
```

Creating a View

- **Creating a View with WITH CHECK OPTION:**

```
CREATE VIEW staff_vw
AS
SELECT *   FROM  staff_master
WHERE  deptno =10 WITH CHECK OPTION constraint cn;
```

Rules for performing operation on View

- **You can perform “DML operations” on simple Views.**
- **You cannot remove a row if the View contains the following:**
  - Group functions
  - A GROUP BY clause
  - The DISTINCT keyword
  - The pseudocolumn ROWNUM keyword

Inline View

- **“Inline view” is not a schema object like a regular View. However, it is a temporary query with an alias.**

```
SELECT dept_name,staff_name,staff_sal
FROM staff_master staff,
(SELECT dept_name,dept_code
 FROM department_master) dept
WHERE staff.dept_code=dept.dept_code
```

Inline View

- **You can use Order By clause, as well, in the Inline View.**
  - This is very useful when you want to find the top n values in a table.

```
SELECT rownum,staff_name
FROM (SELECT staff_name,staff_sal
FROM staff_master ORDER BY staff_sal desc)
WHERE rownum < 5
```

Deleting a Database Objects

Example 2:

If new_emp is a Synonym for a table, then the Table is not affected in any way. Only the duplicate name is removed.

```
DROP SYNONYM new_emp;
```

DBMS/SQL

Set Operators

Lesson Objectives

- **To understand the following topics:**
  - Set Operators
  - UNION operator
  - INTERSECT operator
  - MINUS operator

SET Operators in Oracle

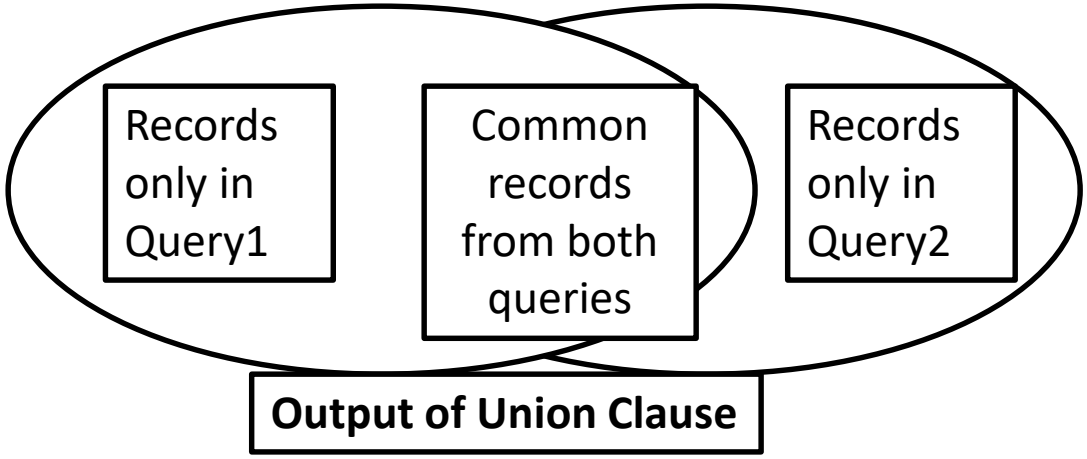
- **SQL supports the following four Set operations:**
  - UNION ALL
    - Combines the results of two SELECT statements into one result set.
  - UNION
    - Same as UNION ALL. Eliminates duplicate rows from that result set.
  - MINUS
    - Takes the result set of one SELECT statement, and removes those rows that are also returned by a second SELECT statement.
  - INTERSECT
    - Returns only those rows that are returned by each of two SELECT statements.

SET Operators in Oracle

- **Each of these operations combines the results of two SELECT statements into a single result.**
- **Note: While using SET operators, the column names from the first query appear in the result set.**

UNION Operator

- **By using the UNION clause, multiple queries can be put together, and their output can be combined.**
- **The UNION clause merges the output of two or more queries into a single set of rows and columns.**



UNION Operator- Example

- **Example: To display all students who are listed for 2006, 2007 and both the years.**

```
SELECT Student_Code
        FROM Student_Marks
        WHERE Student_year=2006
         UNION
SELECT Student_Code
        FROM  Student_Marks
        WHERE Student_year=2007;
```

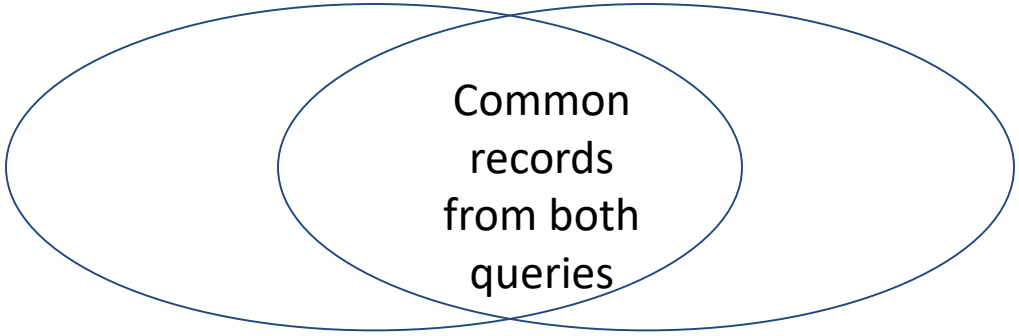
UNION Operator- Example

- **Some situations, if you need duplicate row as well use UNION ALL Operator**

```
SELECT Student_Code
        FROM Student_Marks
        WHERE Student_year=2006
        UNION ALL
SELECT Student_Code
        FROM  Student_Marks
        WHERE Student_year=2007;
```

INTERSECT Operator

- **The INTERSECT operator returns those rows, which are retrieved by both the queries.**



Output of Intersect Clause

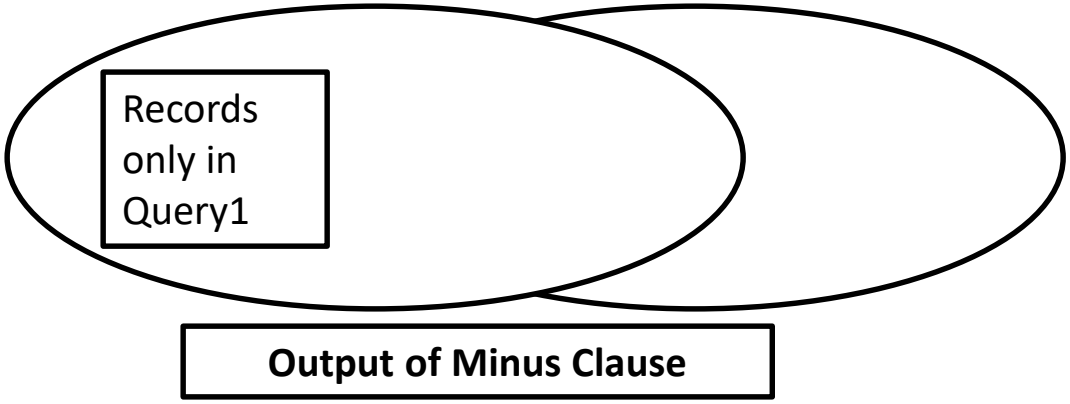
INTERSECT Operator – Example

Example : To display students who are listed for both the years

```
SELECT Student_Code
      FROM Student_Marks
     WHERE Student_year=2006
     INTERSECT
SELECT Student_Code
      FROM Student_Marks
     WHERE Student_year=2007;
```

MINUS Operator

- **The MINUS operator returns all rows retrieved by the first query but not by the second query.**



MINUS Operator - Example

- **Example: To display all students who are listed only for year 2006**

```
SELECT Student_Code
    FROM Student_Marks
    WHERE Student_year=2006
    MINUS
SELECT Student_Code
    FROM Student_Marks
    WHERE Student_year=2007;
```

DBMS/SQL

Data Manipulation Language

Data Manipulation Language

- **Data Manipulation Language (DML) is used to perform the following routines on database information:**
  - Retrieve
  - Insert
  - Modify
- **DML changes data in an object. If you insert a row into a table, that is DML.**
- **All DML statements change data, and must be committed before the change becomes permanent.**

INSERT

- **INSERT command:**
  - INSERT is a DML command. It is used to add rows to a table.
  - In the simplest form of the command, the values for different columns in the row to be inserted have to be specified.
  - Alternatively, the rows can be generated from some other tables by using a SQL query language command.

Inserting Rows into a Table

- **Inserting by specifying values:**

Example: To insert a new record in the DEPT table

```
INSERT INTO table_name[(col_name1,col_name2,...)]
        {VALUES (value1,value2,....) | query};
```

```
INSERT INTO Department_master
VALUES (10, 'Computer Science');
```

Inserting Rows into a Table

- **Inserting rows in a table from another table using Subquery:**

Example: The example given below assumes that a new_emp_table exists. You can use a subquery to insert rows from another table.

```
INSERT INTO new_staff_table
SELECT * FROM staff_master
WHERE staff_master.hiredate > '01-jan-82';
```

Inserting Rows into a Table

- **Inserting by using "substitution variables":**

Example: In the example given below, when the command is run, values are prompted every time.

```
INSERT INTO department_master
VALUES (&dept_code, '&dept_name');
Enter a value for dept_code : 20
Enter a value for dept_name : Electricals
```

DELETE

- **The DELETE command is used to delete one or more rows from a table.**
  - The DELETE command removes all rows identified by the WHERE clause.

```
DELETE [FROM] {table_name | alias }
        [WHERE condition];
```

Deleting Rows from Table

Example 1: If the WHERE clause is omitted, all rows will be deleted from the table.

Example 2: If we want to delete all information about department 10 from the Emp table:

```
DELETE
    FROM staff_master;
```

```
DELETE
    FROM student_master
    WHERE  dept_code=10;
```

UPDATE

- **Use the UPDATE command to change single rows, groups of rows, or all rows in a table.**
  - In all data modification statements, you can change the data in only “one table at a time”.

```
UPDATE table_name
SET   col_name = value|
      col_name = SELECT_statement_returning_single_value|
      (col_name,...) = SELECT_statement
[WHERE condition];
```

Updating Rows from Table

Example 1: To UPDATE the column "dname" of a row, where deptno is 10, give the following command:

```
UPDATE department_master
SET dept_name= 'Information Technology'
WHERE dept_code=10;
```

Updating Rows from Table

Example 2: To UPDATE the subject marks details of a particular student, give the following command:

```
UPDATE student_marks
SET subject1= 80 , subject2= 70
        WHERE student_code=1005;
```

Using a Subquery to do an Update

- **For making salary of "Anil" equal to that of staff member 100006, use the following command:**

```
UPDATE staff_master
SET staff_sal = (SELECT staff_sal FROM staff_master
                                  WHERE staff_code = 100006 )
WHERE staff_name = 'Anil';
```

DBMS/SQL

Transaction Control Language

Defining Transaction

- **A "transaction" is a logical unit of work that contains one or more SQL statements.**
  - "Transaction" is an atomic unit.
  - The effects of all the SQL statements in a transaction can be either:
    - all committed (applied to the database), or
    - all rolled back (undone from the database)
  - A "transaction" begins with the first executable SQL statement.

Defining Transaction

- A “transaction” ends when any of the following occurs:
  - A user issues a COMMIT or ROLLBACK statement without a SAVEPOINT clause.
  - A user runs a DDL statement such as CREATE, DROP, RENAME, or ALTER.
  - If the current transaction contains any DML statements, Oracle first commits the transaction, and then runs and commits the DDL statement as a new, single statement transaction.
  - A user disconnects from Oracle. The current transaction is committed.
  - A user process terminates abnormally. The current transaction is rolled back.

Statement Execution and Transaction Control

- **A "SQL statement" that runs successfully is different from a committed transaction.**
- **However, until the "transaction" that contains the "statement" is** committed**, the "transaction" can be rolled back. As a result, all the changes in the statement can be undone.**
- **Hence we can say, "a statement, rather than a transaction, runs successfully".**

Commit Transactions

- **Committing a transaction means making "permanent" all the changes performed by the SQL statements within the transaction.**
  - This can be done either explicitly or implicitly.

- **Syntax:**

```
COMMIT [WORK];
```

Commit Transactions

- **COMMIT types:**
  - Implicit: Database issues an implicit COMMIT before and after any data definition language (DDL) statement
  - Explicit

Example of COMMIT command:

```
DELETE FROM student_master
WHERE student_name = 'Amit';
COMMIT ;
```

Rollback Transactions

- **Rolling back a transaction means "undoing changes" to data that have been performed by SQL statements within an "uncommitted transaction".**
  - Oracle uses "undo tablespaces" (or rollback segments) to store old values.
  - Oracle also uses the "redo log" that contains a record of changes.
- **Oracle lets you roll back an entire "uncommitted transaction".**
  - Alternatively, you can roll back the trailing portion of an "uncommitted transaction" to a marker called a "savepoint".

Savepoints in Transactions

- **In a transaction, you can declare intermediate markers called "savepoints" within the context of a transaction.**
  - By using "savepoints", you can arbitrarily mark your work at any point within a long transaction.
  - In this manner, you can keep an option that is available later to roll back the work performed, however:
    - before the current point in the transaction, and
    - after a declared savepoint within the transaction
- **For example: You can use savepoints throughout a long complex series of updates. So if you make an error, you do not need to resubmit every statement.**

Examples of Rollback and Savepoints

Example 1:

```
INSERT INTO department_master
VALUES (70, 'PERSONNEL") ;
SAVEPOINT A ;
INSERT INTO department_master
VALUES (80, 'MARKETING') ;
SAVEPOINT B ;
```

```
ROLLBACK TO A ;
```