

Course on Database Management System

Version of database used for demonstrations: Oracle 9i

Faith InfoTech
Thejaswini
Technopark.

Prepared By:
Lily Antony M.

This course would help you get familiar with SQL and the basic RDBMS concepts through a series of problems. If you are not familiar with any of the terms that are mentioned in the explanation, useful links are provided at appropriate areas to guide you to the explanation.

The list of problems that you can use is given below.

1. DATABASE

1. [What is a database?](#)
2. [Why did we have to use a database when we already had register books and file systems to organize our data?](#)
3. [What are the characteristics of a good database?](#)
4. [What advantages do we get if we use a DBMS?](#)
5. [What are the functions of a DBMS?](#)
6. [What are the different types of databases?](#)
7. [What is a data dictionary?](#)
8. [Explain the architecture of the database.](#)
9. [Explain the architecture of the Oracle database.](#)
10. [Explain the different levels of data abstraction.](#)
11. [What advantage do we get through the different levels of data independence?](#)
12. [What are the good practices of SQL query writing?](#)
13. [Why should we have the knowledge of the order in which the query is parsed in a SELECT statement?](#)
14. [Explain the three schema Architecture of the DBMS? How does it lead to efficient computing?](#)
15. We already know the syntax of the SELECT statement and the order of the different clauses in the SELECT statement. [Then why do we need to know the order in which the query is processed?](#)

Evaluation

2. DATA MODELS

1. [How do we use the ER Model?](#)
2. [What are data models and what are the different types of data models?](#)
3. [Why should we use ER Model? Why not explain the design of the tables of the database in simple text, rather than use ER Model?](#)

3. RELATIONAL DATABASE

1. [Explain Relational Model.](#)
2. [What are the features of the relational model?](#)
3. [What is relational algebra?](#)
4. [Why did we need relational model when we already had hierarchical model and network model?](#)

Evaluation

4. NORMALIZATION

1. [Why do we need to normalise the relations? What advantages do we get?](#)
2. [What are the different levels of normalisation?](#)
3. [How do we ensure that tables are in the first normal form?](#)
4. [How do we ensure that tables are in the second normal form?](#)
5. [How do we ensure that tables are in the third normal form?](#)
6. [To what level should we normalize?](#)
7. [What is BCNF?](#)
8. [How do we ensure that a table is in BCNF?](#)
9. [How do we achieve the fourth normal form?](#)
10. [What is fifth level of normalization?](#)
11. [What is de-normalisation?](#)
12. Problem for Third normal form with School database
13. Problem for Third Normal form with Retail store database
14. Problem of Second Normal form with Project Management database.

15. [Why do we need to de-normalise?](#)
16. [If majority of the applications work fine with 3NF, why do we need to go for BCNF?](#)

Evaluation

5. DDL and DCL

1. [What are DDL statements?](#)
2. [What are DCL statements?](#)

Evaluation

6. CREATE

1. [How do we create a table?](#)
2. How do we establish relationship between two tables in the relational model? (problem)
3. How do you create a table having the same structure as the existing table and having all the data of the table? (problem)
4. How do we create a table with same structure but with different column names as the existing table and without copying the data from the existing table?
5. How to create a copy of an existing table by including only some of the columns of the existing table?
6. How do we add UNIQUE and NOT NULL constraints to a table? (problem)
7. How do we set a default value to a column of a table? (problem)
8. A comprehensive problem to create a table.

7. ALTER

1. How do we change the data type of the column of a table? (Problem)
2. How do we remove the DEFAULT value set for a column? (Problem)
3. How do we set primary key and foreign Key for an existing table? (Problem)
4. How to add a Check Constraint to an existing table? (Problem)
5. How do we remove the check constraint set on the column of a table? (Problem)
6. How to set the column of an existing table as Not Null? (Problem)
7. How do we remove the column of a table (Problem)
8. How do we add a column to an existing table? (Problem)
9. Allow only values 'SSLC', 'Plus2', 'P.D.C.', 'B.Tech.', 'M.C.A.' to the Qualification name in the Qualification table of Project Management database.

8. DROP

1. How do we remove a table from the database? (Problem)
2. How do we remove the primary key constraint set on the column of a table? (Problem)
3. How do we remove the foreign key constraint set on the column of a table? (problem)

9. RENAME

1. How do you rename an object/ table in the database(problem)
2. How to rename the column of a table?

10. TRUNCATE

1. How do you remove all the data from a table without firing any triggers? (problem)

11. CONSTRAINTS

1. [What are constraints?](#)
2. [Why do we need constraints?](#)
3. How do we set the column of a table as primary key while creating the table? (problem)
4. How do we set a restriction that only a specified set of values can be inserted into a column of a table? (problem)
5. How to set a column of an existing table as foreign key(problem)
6. How to remove a constraint(problem)
7. How to remove the child rows when the parent rows are deleted? (problem)

Evaluation

12. DML

1. [What is DML?](#)

Evaluation

13. INSERT

1. How do you add values to a table?
2. How do you copy the data in a table to another table with a single command?
3. Add values to the ProductID and Product name only for the Product table.
4. Add the id and name of all staff that has BTech to a 'HighQual' table.

14. [UPDATE](#)

1. How do you change the existing data?
2. Update the total score column with the total score for each candidate?
3. Change the name of all products by adding 'X' at the end of the name of all products that has price lower than 20.
4. Change the name of all schools that begin with ABC to XYZ International School in the Qualification table of project management.

15. [DELETE](#)

1. How do we remove the data of all students whose name begins with 'A' and Total score is below 200?
2. How do we remove all the students who were born in the year 98?
3. How do we remove the data of all the batches that are not following Syllabus S20?
4. Remove the skills that have second letter as 'C';

16. [SELECT](#)

1. How do we retrieve the data in the format "Average mark of *StudentID* is 80" for all the students? The data should come under the heading 'Names and Percentages'.
2. How do we retrieve the name and ID of all the students who are born in the month of January? The Student Name column should have the heading "Name" and ID should have the heading "Student Code". Add a serial number at the beginning.
3. How do we retrieve the ID of all students who have scores between 400 and 450? Include these two scores in the search.
4. How do we retrieve the all the marks, minimum mark and percentage for each student. Assume that the maximum score for each subject is 100?
5. Retrieve all the data in the Category table.

17. [ORDER BY](#)

1. How do we retrieve the names and marks of all students in the descending order of marks? For students having the same marks the details should be displayed in the ascending order of name. The first column should be SerialNumber with each row having serial number 1, 2, 3 etc.
2. [Can we have an ORDER BY clause with the column that is not specified in the SELECT clause?](#)
3. What is the meaning of the statement SELECT StudentName FROM Student Order BY 1;?
4. How do we retrieve the student names and date of birth in the ascending order of their age?
5. To assign to a project we need to find staff with Java and database skills. How do we do this? Sort in ascending order of skills
6. To assign to a project we need to find staff with at least one of the skills Java or database skills. How do we do this? Sort in descending order of skills

18. [JOIN](#)

1. [Why did we need the concept of Joins?](#)
2. [What are the different types of Joins?](#)
3. [What is a CROSS JOIN?](#)
4. [What is the difference between the Equi Join and Natural Join?](#)
5. [How do we retrieve details from more than two tables?](#)
6. [Explain Self Join.](#)
7. How do we retrieve the names of all students and NCC rank for only those students who are part of NCC?
8. How do we retrieve the names and ages of all students and NCC rank irrespective of whether the student is part of NCC Group?

9. How do we retrieve the name of the student and the grade in academics based on the total annual score? Retrieve the data as "StudentName has # % ". Sort the data in the descending order of the score. Include only the details of those students who are in Batch B1 and whose scores are in the range 200 to 250.
10. How do we retrieve the names of students and their team leaders? Also include the team leaders themselves who have no team leaders. Sort the data in the ascending order of the student's name.
11. How do we retrieve the names of all the students who are in Standard 1 batch A and following CBSE Syllabus?
12. How do we find the name of the products that have reached the reorder level?
13. Product name and Order details of the product 'Mavila'

Evaluation

19. SUB QUERIES

1. [What is a Subquery?](#)
2. [What is the need of subquery when we already have Joins?](#)
3. [What are the different types of subqueries?](#)
4. [What is the behaviour of the correlated sub-query?](#)
5. [What is the special feature of the scalar sub-query?](#)
6. How do you retrieve the nth largest element from a table?
7. How we retrieve the details of all the students who are in having the same total score as 'Tony George'. Do not include details of 'Tony George' in the result.
8. How do we retrieve the name of the batch that has the maximum number of students?
9. How do we retrieve batchname, number of students for each batch?
10. How do we retrieve details of the students who earn first three positions for Batch B1?
11. How do we display the scores of all the team leaders?
12. How do we display only the odd numbered records?
13. Retrieve the work sheet of Tony from the project management database.

Evaluation

20. FUNCTIONS

1. [What are the different inbuilt functions available in Oracle?](#)
2. How do we find the age of all the students who were born after 1989?
3. How do we find the class name and the number of students in each batch?
4. How do we display the names in Sentence case and total marks of all candidates in the format 1,234.89?
5. How do we display the date of birth of all the candidates in the format "first of June, 1999"?
6. How do we display the date of birth of all the candidates in the format " June One, Nineteen Ninety Nine 10:25 pm"?
7. How do we display the date of the next Monday that fall after today in the format 'Monday, August 30, 98'?
8. How do we display the batch name, name short forms and age of all the students in the ascending order of their age? Display only the details of students who are older than 12.
9. How do we display pass for all students with marks 100 and 149 distinction for marks 250 and above and failed for marks below 100?
10. How do we find a unique code for all the students by concatenating the student ID, first three letters of the name in upper case their batch name and standard name?
11. How do we find the name of the student their marks for the different subjects and the highest score from among their scores for different subjects? Sort the result in the descending order of the highest score. For students having same highest scores sort them in the ascending order of the names.
12. How do we create a list of company name short forms? Short forms are created by taking the first three letters of the company name.
13. How do we retrieve the prices of all the tooth care products rounded to the nearest 10th place? Give the column heading as "Approximate price". Sort it in the ascending order of the price.

14. How do we retrieve the price of all products costlier than 1000 truncated to the nearest 100? The output should be in the format \$1,200.00 under the heading "Price after Discount"
15. The retail store wants to develop a code list for the company names by replacing 'C' with 'D'; 'A' with 'B' and 'P' with 'Q'. How do we do this?
16. How do we retrieve the category name, Company name, product name and the price?
17. How do we retrieve the name and salary of the first three senior most staff in the company?
18. How many projects has the company done in the Finance domain?
19. Domains that have projects longer than 2 months.

Evaluation

21. GROUP BY

1. [Why do we have a HAVING clause when we already have a WHERE clause for filtering data?](#)
2. [What are the features of the GROUP BY clause?](#)
3. [What are Group functions?](#) How are they related to GROUP BY clause?
4. How do we find the batch name and the maximum score in each class?
5. How do we retrieve the number of students in each batch? Sort the data in the descending order of the strength.
6. How do we retrieve the date of births and the number of students sharing the same date of births? Sort the data according to the dates of birth.
7. How do we find the total quantity of the different products (along with the product names) that has been ordered so far, with the date on which the last order was made? Display the data in the descending order of the quantity.
8. How do we find the number of departments in each floor?

Evaluation

22. Transaction and TCL

1. [What is a transaction?](#)
2. [What is TCL?](#)
3. [What are the features of TCL?](#)
4. [How do we rollback to a specific point without affecting all the changes made?](#)
5. [What is RollBack?](#)
6. [What is rolling forward?](#)
7. [What is concurrency control in DBMS? Why do we need it and what are the different methods of concurrency control in DBMS](#)
8. [What are locks? How does it work? What are the Different types of locks?](#)
9. [What is a deadlock?](#)
10. [What is the ACID rule for database transactions?](#)

Evaluation

23. Performance

1. [What is performance tuning?](#)
2. [What are the different ways optimising a query to increase its performance?](#)

Evaluation

Sample Database

The problems discussed are based on the following database table

1. Primary school.
2. Retail Store
3. Hardware Company

The script to create the tables without any constraints is given at the end of this database.

Primary School**Syllabus**

SyllabusID	SyllabusName	Description
S10	CBSC	Central School Syllabus
S20	ICSE	ICSE Syllabus
S30	SSLC	State Syllabus

Standards

StandardID	StdName	SyllabusID
T1	I	S10
T2	II	S10
T3	III	S10
T4	IV	S10
T5	I	S20
T6	II	S20
T7	III	S20
T8	IV	S20

Batches

BatchID	BatchName	StandardID
B1	A	T1
B2	B	T1
B3	A	T2
B4	B	T2
B5	A	T3
B6	B	T3
B7	A	T4
B8	A	T1
B9	A	T2
B10	A	T3
B11	A	T4

Student

StudentID	TeamLeaderID	StudentName	DateofBirth	BatchID
ST1		Mathew Haiden	01-Jan-97	B1
ST 2	ST2	Mary Claire	28-Mar-96	B1
ST 3	ST2	Muhammed Ali	15-Jun-95	B1
ST 4	ST5	Elton John	11-Sept-98	B2
ST 5		Hanna Montanna	31-Dec-98	B2

Marks

StudentID	StdID	Mark1	Mark2	Mark3	Total
ST1	T1	80	75	78	
ST 2	T1	90	81	76	
ST 3	T1	61	78	88	
ST 4	T1	71	81	81	

ST 5	T1	85	68	80	
------	----	----	----	----	--

NCC

StudentID	Grade	Rank
ST1	I	A
ST 2	I	A
ST 3	II	B

Script to create the tables

```
CREATE TABLE Syllabus
(
    SyllabusID      Varchar2(3),
    SyllabusName    Varchar2(30),
    Description      Varchar2(30)
);
CREATE TABLE Standard
(
    StandardID      Varchar2(3),
    StdName         Varchar2(10),
    SyllabusID      Varchar2(3)
);
CREATE TABLE Batch
(
    BatchID         Varchar2(3),
    BatchName       Varchar2(2),
    StandardID      Varchar2(3)
);
CREATE TABLE Student
(
    StudentID       Varchar2(6),
    TeamLeaderID    Varchar2(6),
    StudentName     Varchar2(30),
    DateOfBirth     Date,
    BatchID         Varchar2(3)
);
CREATE TABLE Mark
(
    StudentID       Varchar2(6),
    StdID           Varchar2(3),
    Mark1           Number(3),
    Mark2           Number(3),
    Mark3           Number(3),
    Total           Number(3)
);
CREATE TABLE NCC
(
    StudentID       Varchar2(6),
    Grade           Varchar2(2),
    Rank            Varchar2(2)
);
```


Retail Store

This is the relational model representation of a retail store. The retail store has products in different categories and these products belong to different companies. When the products go below reorder level, the person in charge places the order for the products. The details of the orders placed are in the Order table.

The script to create these tables without any constraints is given at the end of the database.

Company Table:

Code	Company	Name
112232		Colgate Palmolive
112233		PepsiCo
112234		Coco cola India
112235		Dhathri
112236		Meswak India
112237		Hindustan Lever

Category

CategoryID (#)	CatName	CompanyID
11	ToothCare	112232
12	Skin Care	112235
13	Hair Care	112235
14	Health Care	112237
15	Skin Care	112237
16	Hair Care	112237
17	Food	112233

Product Table:

Product ID (#)	Name	Price	Weight gms	Category ID (FK)
11290	Colgate Active Salt	45.95	200	11
11291	Closeup Cool crystals	48.20	200	11
11292	Pepsodent	40.10	200	11
11293	Meswak	32.35	200	11
11294	Mavila	35	200	11
11295	Lays	10	100	17
11296	Abc	1256.30	0	12

Stock

Product ID	Quantity	ReorderLevel
11290	1000	200
11291	1500	200
11292	2000	100
11293	3000	300
11294	2000	200
11295	1000	150

Order

OrderDate	ProductID	Quantity
1-Jan-99	11293	500
21-Jun-99	11294	500
23-Sept-99	11295	500
23-Sept-99	11294	500
23-Sept-99	11293	200
30-Nov-99	11295	500

Script

```
CREATE TABLE Company
(
    CompanyCode Number(6),
    Name        Varchar2(30)
);
CREATE TABLE Category
(
    CategoryID   Number(3),
    CatName      Varchar2(30),
    CompanyID    Number(6),
);
CREATE TABLE Product
(
    ProductID    Number(5),
    Name         Varchar2(30),
    Price        Number(7,2),
    Weight       Number(5,2),
    CategoryID   Number(3)
);
CREATE TABLE Stock
(
    ProductID    Number(5),
    Quantity     Number(4),
    ReorderLevel Number(4),
);
CREATE TABLE Order
(
    ProductID    Number(5),
    OrderDate    Date,
    Quantity     Number(4)
);
```

Project Management

This is the relational model representation of a project management application. The details of the staff of the company are stored in the Staff table. The details of the project leader and team members are in a team table. The details of the qualification and the technical skills of the staff are stored in the Qualification and Skill tables. The details of work allotment are stored in the WorkAllotment table.

The script to create the tables in the database without constraint is given at the end of this database.

Department

Dept ID (#)	Name	Location
10	Account	Floor 1
11	Sales	Floor1
13	IT	Floor2
14	System	Floor2

Staff

StaffID (#)	Name	Salary	DateOfJoin	DeptID (FK)
100	Tony	45000	02-APR-98	13
101	Lily	55000	10-APR-02	13

Qualification

QualID	StaffID (FK)	QualificationName	Institute	YearofCompletion	Percentage
1	100	SSLC	ABC School	1986	90
2	100	Plus2	ABC School	1988	90
3	100	BTech	XYZ Institute of Technology	1992	78

Skills

StaffID	SkillName	Years
100	C	1
100	C++	1
100	Java	2
100	Database	2.5

ProjectDomain

DomainID (#)	Name	Description
1	Finance	Finance
2	Retail	Retail

Project

ProjectID	Name	DurationDays	Description	StartDate	EndDate	DomainID
100	XZ Retail	100	Intermediate	10-APR-02	10-JUN-02	2
101	YZ Financiers	150	Complex	10-JUN-06	10-DEC-06	1

Work Allotment

Project ID	StaffID	ProjectLeaderID
100	100	100
100	101	100

WorkCategory

WCatID	Name
1	Project
2	Presentation
3	Upgrade

WorkSheet

WorkID	StaffID	WorkDateTime	WorkCatID	DurationMinutes
1	100	10-APR-02 10:20	1	100
2	100	10-APR-02 2:20	2	100

Script

```
CREATE TABLE Department
```

```
(      DeptID      Number(3),  
      Name         Varchar2(30),  
      Location     Varchar2(30)
```

```
);
```

```
CREATE TABLE Staff
```

```
(      StaffID Number(3),  
      Name      Varchar2(30),  
      Salary    Number(6),  
      DateOfJoin Date,  
      DeptID    Number(3)
```

```
);
```

```
CREATE TABLE Qualification
```

```
(      QualID      Number(5),  
      StaffID      Number(3),  
      QualName     Varchar2(30),  
      Institute     Varchar(100),  
      YearOfCompletion Number(4),  
      Percentage    Number(5,2)
```

```
);
```

```
CREATE TABLE Skill
```

```
(      StaffID      Number(3),  
      SkillName     Varchar2(30),  
      Years         Number(2)
```

```
);
```

```
CREATE TABLE ProjectDomain
```

```
(      DomainID    Number(3),  
      Name         Varchar2(30),  
      Description   Varchar2(100)
```

```
);
```

```
CREATE TABLE ProjectDomain
```

```
(      ProjectID    Number(3),  
      Name          Varchar2(30),  
      Duration       Number(4),  
      Description    Varchar2(100),  
      StartDate     Date,  
      EndDate       Date,  
      DomainID      Number(3)
```

```
);
```

```
CREATE TABLE WorkAllotment
```

```
(      ProjectID    Number(3),  
      StaffID       Number(3),
```

```
        ProjectLeaderID      Number(3),
    );
CREATE TABLE WorkCategory
(
    WCatID      Number(3),
    Name        Varchar2(30)
);
CREATE TABLE WorkSheet
(
    WorkID      Number(3),
    StaffID     Number(3),
    WorkDateTime Date,
    WCatID      Number(3),
    Duration    Number(4)
);
```

Problem

5.11 Problem for Third Normal form with School database

Consider the relations given below. Are they in the third normal form?

StdID	StdName	Syllabus
T1	I	CBSE
T2	II	CBSE
T3	III	CBSE
T4	IV	CBSE
T5	I	ICSE
T6	II	ICSE
T7	III	ICSE
T8	IV	ICSE

Solution

The rule for third normal form is that the attributes should not have transitive dependencies. That is all attributes should be dependent only on the primary key. If there is any other dependency it should be removed to another table.

Here the dependencies are:

StandardName → StdID

Syllabus, StandardName → StandardID

So there is no violation of the third normal form.

5.12 Problem of Third Normal form with the Retail Store database

Consider the relation. Is it in Third normal form?

Category

ProductID (#)	ProductName	CatName
1	XYZ	ToothCare
2	AB	Skin Care
3	CP	Hair Care
4	CD	Health Care
5	AC	Skin Care
6	XP	Hair Care
7	YZ	Food

Solution

Here the dependencies are

ProductName -> ProductID

CatName -> ProductName

CatName->ProductID

Here the defects are:

- The category name has to be repeated for every product.
- If the only product in a category is removed, the detail that there is such a category is there is also removed.
- If a category name is to be changed, the change should reflect in all the rows of that category.

So there is redundancy, update anomaly and delete anomaly.

Here dependencies other than to primary key should be removed to another table.

So the structure of the table becomes,

Product

ProductID (#)	ProductName	CatName
1	XYZ	1
2	AB	2
3	CP	3
4	CD	4
5	AC	2
6	XP	3
7	YZ	5

Category

CatID	CatName
1	ToothCare
2	Skin Care
3	Hair Care
4	Health Care
5	Food

Now the tables are in [Third Normal form](#).

5.13 Problem of Second Normal Form with Project Management database.

Consider the table

Skills

StaffID	SkillName	YearsOfExperience
100	C	1
100	C++	1
100	Java	2
100	Database	2.5

Is it in the 2NF and 3NF?

Solution

The rule for 2NF is that all the attributes should be functionally dependent on the whole primary key.

Here the primary key is {StaffID, SkillName}. The attribute Years of Experience is functionally dependent on the whole primary key. So it is in 2NF.

The rule for 3NF is that there should not be any transitive dependencies. The attribute Years of Experience does not have any dependency other than to the primary key. So it is in 3NF.

5.14 Consider the table. Convert it to 5NF.

StdID	StdName	Syllabus
T1	I	CBSE
T2	II	CBSE
T3	III	CBSE
T4	IV	CBSE
T5	I	ICSE
T6	II	ICSE
T7	III	ICSE
T8	IV	ICSE

Solution

To convert this relation to 5NF, break it into relations without loss of data.

So the tables are:

Standard

StdID	StdName	Syllabus
T1	I	S1
T2	II	S1
T3	III	S1
T4	IV	S1
T5	I	S2
T6	II	S2
T7	III	S2
T8	IV	S2

Syllabus

Syllabus	Syllabus
S1	CBSE
S2	ICSE

6.2 How do we [establish relationship between two tables](#) in the relational model?

We need to create the Syllabus Table and the Standard Table. The Syllabus table is the parent table with SyllabusID as the Primary Key. The StandardID is the primary key and the SyllabusID is the foreign key in the Standard table referring to the SyllabusID in the Syllabus Table. The relational model representation of the tables is given.

Syllabus

SyllabusID	SyllabusName	Description
S10	CBSC	Central School Syllabus
S20	ICSE	ICSE Syllabus
S30	SSLC	State Syllabus

Standards

StdID	StdName	SyllabusID
T1	I	S10
T2	II	S10
T3	III	S10
T4	IV	S10
T5	I	S20
T6	II	S20
T7	III	S20
T8	IV	S20

How do we create the Syllabus Table and the Standard Table with the parent child relationship?

Solution

To solve this problem we use the Primary Key and Foreign Key constraints.

To identify the [data types](#), we can look at the sample data. All the columns of the Syllabus table and hold character string data.

To create the Syllabus Table

```
CREATE TABLE Syllabus
(
    SyllabusID    Varchar2(3)    PRIMARY KEY,
    SyllabusName  Varchar2(30),
    Description    Varchar2(100)
);
```

To create the Standard Table we can use the following Syntax. Here I have opted to use the [column level](#) syntax. You can also use the [table level](#) syntax since the effect of both are the same.

```
CREATE TABLE Standard
(
    StandardID    Varchar2(4)    PRIMARY KEY,
    StandardName  Varchar2(10),
    SyllabusID    Varchar2(3)    REFERENCES Syllabus(SyllabusID)
);
```

Please make sure that the data type including the size of the type of the Primary Key and the foreign Key column should be the same.

6.3 How do you create a table having the same structure as the existing table and having all the data of the existing table?

We need to create a copy of the Syllabus Table named SyllabusCopy which has the same structure and data as the syllabus table. How would we do that?

Solution

To create a table having the same structure and data as an existing table we need to use the [CREATE statement along with a sub query](#).

```
CREATE SyllabusCopy  
AS SELECT * FROM Syllabus;
```

6.4 How do we create a table with same structure but with different column names as the existing table and without copying the data from the existing table

The copy of the Syllabus table named SyllabusCopy should have the column names SyllabusCode, Name and remarks instead of the original column names and it should not have any data. How would we solve this?

Solution

To create a table with the same structure as the existing table we need to use the [CREATE statement with a subquery](#) which is a SELECT statement.

The data is also copied to the new table since the SELECT statement retrieves the data from the existing table. To prevent the SELECT statement from retrieving data, we need to add a WHERE clause that has a condition that always evaluates to true.

To give the columns new names in the table copy, we can either mention the new column names in the CREATE clause like this

```
CREATE TABLE SyllabusCopy (SyllabusCode, Name, Remarks)
AS SELECT * FROM Syllabus;
```

OR mention the new column names as [column alias](#) in the SELECT Clause.

```
CREATE TABLE SyllabusCopy
AS SELECT SyllabusID AS SyllabusCode, SyllabusName AS Name, Description AS Remarks FROM Syllabus;
```

Here though the column names are different, the datatypes would be automatically taken from the existing table.

6.5 How to create a copy of an existing table by including only some of the columns of the existing table?
We need a [copy of the existing Syllabus Table](#), but the copy of the table need to include only SyllabusID and SyllabusName. The name of the SyllabusName column in the new table should be 'Name'.

Solution

To create a copy of the table we need to use the [CREATE statement with the subquery](#). But a simple subquery will copy the data also. The data is also copied to the new table since the SELECT statement retrieves the data from the existing table. Since we are including only some of the columns in the new table explicitly mention the column names in the SELECT clause.

To give the columns new names in the table copy, we can either mention the new column names in the CREATE clause like this

```
CREATE TABLE SyllabusCopy (SyllabusCode, Name)  
AS SELECT SyllabusID, SyllabusName FROM Syllabus;
```

OR mention the new column names as [column alias](#) in the SELECT Clause.

```
CREATE TABLE SyllabusCopy  
AS SELECT SyllabusID AS SyllabusCode, SyllabusName AS Name FROM Syllabus;
```

6.6 How do we add Unique and Not null constraints to a table?

In the Syllabus table we need to make the Syllabus description a mandatory column. The Syllabus table holds the information of the various syllabi. We also need to make sure that one syllabus name is mentioned only once.

Solution

To make a column mandatory, we need to add the Not null constraint and to ensure that there is no repetition in the data in a column we add the unique constraint.

If we need to add the constraint during the creation of the table, we need to do this using the CREATE statement.

```
CREATE TABLE Syllabus
(
    SyllabusID      Varchar2(3)    PRIMARY KEY,
    SyllabusName    Varchar2(30) UNIQUE,
    Description     Varchar2(100) NOT NULL
);
```

Suppose that the Syllabus table already exists and the constraints are to be added to the existing Syllabus table. Then we need to add the constraints using the ALTER statement.

```
ALTER TABLE Syllabus ADD CONSTRAINT Syllabus_Unq UNIQUE(SyllabusName) MODIFY Description
Varchar2(100) NOT NULL;
```

Here the CONSTRAINT clause giving a [constraint name](#) to the UNIQUE constraint is optional.

The unique constraint is added using the ADD clause.

NOT NULL is the only constraint that can be added using the MODIFY constraint.

6.7 How do we set a default value to a column of a table?

The Syllabus Table has the description column, which should take a default value 'Syllabus' if it is no value is given for the column.

How will you solve this?

Solution:

For setting a default value for a column we should specify it when we create the table or add it later using the ALTER statement.

If we want to set the default value for a column while we create the table, we can use this syntax.

```
CREATE TABLE Syllabus
(
    SyllabusID      Varchar2(3)    PRIMARY KEY,
    SyllabusName    Varchar2(30) UNIQUE,
    Description      Varchar2(100) NOT NULL          DEFAULT 'Syllabus'
);
```

If we want to set the default value for an existing table, the use this syntax:

```
ALTER TABLE Syllabus MODIFY Description Varchar2(100) NOT NULL          DEFAULT 'Syllabus';
```

Here we used the MODIFY clause because we are modifying the existing column Description to set the default value.

10.1 How do we change the data type of the column of a table?

In the Syllabus table, the description table has the data type varchar2 with size 100. We need to reduce the number of characters that can be stored to 50.

Solution

To make modification to the existing columns, we need to use the ALTER statement with MODIFY clause.

We need to specify all the

```
ALTER TABLE Syllabus MODIFY Description varchar2(50);
```

IMPORTANT: Please keep in mind that this syntax does not change any earlier constraints or default values that were set to the column earlier.

10.2 How do we remove the DEFAULT value set for a column?

A DEFAULT value of 'Syllabus' was set to the Description column. We need to remove the DEFAULT setting for this column.

Solution

The DEFAULT value that is set for a column cannot be removed. But it can be reset to NULL using the DEFAULT clause. See what the Oracle documentation has to say about this.

"If a column has a default value, then you can use the DEFAULT clause to change the default to NULL, but you cannot remove the default value completely. That is, if a column has ever had a default value assigned to it, then the DATA_DEFAULT column of the USER_TAB_COLUMNS data dictionary view will always display either a default value or NULL"

10.3 How do we set primary key and foreign Key for an existing table?

The Standard table was set as created without adding constraints. Now we need to set the StandardID as the Primary Key and the SyllabusID as the foreign key referring to the Syllabus ID column of the Syllabus table, which is the primary key for the table.

How do we add a primary key and foreign key?

Solution

To add a constraint to an existing table, we need to use the [ALTER](#) statement with the ADD clause. Both the constraints can be added to the table using the same ALTER statement.

```
ALTER TABLE Standard ADD PRIMARY KEY (StandardID)
                        ADD FOREIGN KEY (SyllabusID) REFERENCES Syllabus(SyllabusID);
```

10.4 How to add a Check Constraint to an existing table?

The StdName column should take only the values 'I', 'II', 'III' or 'IV'. How do we implement this?

Solution

To restrict the values that go into a column to a set of values, we need to use the CHECK constraint. Since we need to add the CHECK constraint to the column of an existing table, we need to use the ALTER statement with the ADD clause.

```
ALTER TABLE Standard ADD CONSTRAINT Std_Check CHECK (StdName IN ('I', 'II', 'III', 'IV'));
```

The effect of the query would be that only 'I', 'II', 'III' or 'IV' would be allowed into the column. The IN clause is used as the comparison operator since the value being inserted is compared to a set of values.

10.5 How do we remove the check constraint set on the column of a table?

We need to remove the CHECK constraint that is set for the StdName column of the Standard table.

Solution

To remove a constraint of a table, we need to use the DROP clause with the [constraint name](#) in the ALTER statement. If you do not remember the constraint name we can find it by querying the user_constraints view, which is part of the [data dictionary](#).

```
SELECT constraint_name, constraint_type  
FROM user_constraints WHERE table_name= 'STANDARD';
```

When you query the user_constraints view make sure that the table name in the WHERE clause is given in upper case since that is how the names of the tables are stored in the data dictionary. Remember that data in the database is case sensitive.

You can identify the check constraint from the constraint type. The constraint type for the check constraint is 'C'.

```
ALTER TABLE Standard DROP CONSTRAINT Std_check;
```

10.6 How to set the column of an existing table as NOT NULL?

We need to make the description column of the Syllabus table as a mandatory.

Solution

Not null is the only constraint that can be added by using the MODIFY clause in the ALTER statement.

```
ALTER TABLE Syllabus ADD Description Varchar2 (100) NOT NULL;
```

If you try the syntax by which other constraints are added, that is by using the ADD clause, you would get an error.

```
ALTER TABLE Syllabus ADD not null(col3);
```

*

ERROR at line 1:

ORA-00904: : invalid identifier

10.7 How do we remove the column of a table?

We need to remove the Description column from the Syllabus table.

Solution

To remove a column we need to use the ALTER statement with the DROP clause

```
ALTER TABLE Syllabus DROP COLUMN Description;
```

10.8 How do we add a column to an existing table?

Add a column named Remarks to the Syllabus table, which can takes some remarks up to 100 characters on each syllabus. It should be a mandatory column.

Solution

To add a column to the existing table we need to use the ALTER statement with the ADD clause. Since it takes character strings, the data type can be varchar2 with size 100. To make the clomn mandatory w use the constraint NOT NULL.

```
ALTER TABLE Syllabus ADD Remarks varchar2(100) NOT NULL;
```

10.9 Allow only values 'SSLC', 'Plus2', PDC', 'BTech', MCA' to the Qualification name in the Qualification table of Project Management database.

Solution

To restrict the values that go into a column to a specific set, we use the CHECK constraint. To add the CHECK constraint to an existing table, we use the ALTER statement with the ADD clause.

We use this syntax

```
ALTER TABLE Qualification
```

```
ADD CONSTRAINT Qual_Check
```

```
CHECK (QualName IN(values 'SSLC', 'Plus2', PDC', 'BTech', MCA'));
```

We used the IN operator here to compare the value added to the column to a set of values. If there is a match the value is accepted into the attribute, else a CHECK constraint violation error is displayed.

9.1. How do we remove a table from the database?

We do not need the SyllabusCopy table anymore. How do we remove it from the database?

Solution

To remove an object from the database, we use the [DROP statement](#).

```
DROP TABLE SyllabuCopy;
```

9.2. How do we remove the primary key constraint set on the column of a table?
We need to remove the primary key constraint of the Standard table.

Solution

To remove a constraint of a table, we need to use the DROP clause with the [constraint name](#) in the ALTER statement. If you do not remember the constraint name we can find it by querying the user_constraints view, which is part of the [data dictionary](#).

```
SELECT constraint_name, constraint_type  
FROM user_constraints WHERE table_name= 'STANDARD';
```

When you query the user_constraints view make sure that the table name in the WHERE clause is given in upper case since that is how the names of the tables are stored in the data dictionary. Remember that data in the database is case sensitive.

You can identify the check constraint from the constraint type. The constraint type for the Primary key constraint is 'P'.

```
ALTER TABLE Standard DROP CONSTRAINT SYS00023;
```

9.3. How do we remove the Foreign key constraint set on the column of a table?

We need to remove the Foreign key constraint of the Standard table.

Solution

To remove a constraint of a table, we need to use the DROP clause with the [constraint name](#) in the ALTER statement. If you do not remember the constraint name we can find it by querying the user_constraints view, which is part of the [data dictionary](#).

```
SELECT constraint_name, constraint_type  
FROM user_constraints WHERE table_name= 'STANDARD';
```

When you query the user_constraints view make sure that the table name in the WHERE clause is given in upper case since that is how the names of the tables are stored in the data dictionary. Remember that data in the database is case sensitive.

You can identify the check constraint from the constraint type. The constraint type for the Foreign key constraint is 'R'.

10. ALTER TABLE Standard DROP CONSTRAINT SYS00024;

9.1 How do you rename an object/ table in the database?

We want to rename the SyllabusCopy table as Syllabus_BU

How can we do this?

Solution

We can use the RENAME statement like this.

```
RENAME SyllabusCopy TO Syllabus_BU
```

11. How do you remove all the data from a table without firing any triggers?

The Syllabus table has a trigger that fires on deleting the records. But, we need to remove all the data from the Syllabus table without firing any triggers. The data that is removed need not be recovered by rollback.

How can we do this?

Solution

To remove the data from the table without firing any triggers, then we can remove the entire data in the table using the [TRUNCATE statement](#)

```
TRUNCATE TABLE Syllabus;
```

11.6 How to remove the child rows when the parent rows are deleted?

We need the details of the students in a batch to be removed when a row in the batch is deleted. Relationship is established between the two tables by BatchID which is primary key in the Batch table and foreign key in the Student table.

How do we do this?

Solution

To remove corresponding rows in the child table, when the parent table rows are removed, we need to use the [ON DELETE CASCADE constraint](#) along with the Foreign key constraint.

For this first remove the existing foreign key and then add a foreign key along with ON Delete Cascade.

```
ALTER TABLE Student DROP CONSTRAINT Std_frqnKey;
```

```
ALTER TABLE Student ADD CONSTRAINT Std_frqnKey  
FOREIGN KEY(BatchID) REFERENCES Batch(BatchID) ON DELETE CASCADE;
```

13.1 How do you add values to a table?

We need to add the details of two Syllabus to the Syllabus table. We need to fill only the ID and the SyllabusName since the Description already has a default value 'Syllabus' set.

How do we do this?

Solution

We use the INSERT statement to add data to the table. Each row is added by a separate INSERT statement.

We can use this syntax which is explicit and straight forward.

```
INSERT INTO Syllabus (SyllabusID, SyllabusName) VALUES (S10, 'CBSE')
```

```
INSERT INTO Syllabus (SyllabusID, SyllabusName) VLAUES (S20, 'ICSE');
```

Since we are adding data to the first two columns of the table we may also omit the column names.

```
INSERT INTO Syllabus VALUES (S10, 'CBSE')
```

```
INSERT INTO Syllabus VLAUES (S20, 'ICSE');
```

Though both syntax would work. But it is always a good practice to use the first syntax since there is no chance of error due to change in the structure of the table unless the change is to the two columns used in the statement.

13.2 How do you copy the data in a table to another table with a single command?

We need to copy the data of the Syllabus table to the SyllabusCopy table that does not have any data.

Solution

To copy all the data from an existing table to another existing table of the same structure we can use [INSERT with SUBQUERY](#)

```
INSERT INTO SyllabusCopy SELECT * FROM Syllabus;
```

13.3 Add values to the ProductID and Product name only for the Product table.

Solution

To add rows to a table we use the [INSERT](#) statement. Since we do not need to add values to all the columns of the table we can mention the names of the columns to which the data is to be inserted.

```
INSERT INTO Product (ProductID, ProductName) VALUES (101, 'TestProduct');
```

This syntax would work even if the column names are not mentioned since the product id and the product name are the first two columns of the table and they appear in the same order in the table also. But it is a good practice to explicitly mention the names of the columns to which you insert the values rather than rely on the order of the columns.

13.4 Add the id and name of all staff that has B.Tech. to a 'HighQual' table. Assume that HighQual is an existing table with columns ProductID and ProductName.

Solution

Here we have to transfer a part of the data that is there in one table to another table. For this we use an

[INSERT](#) statement with a subquery.

The subquery is a [SELECT](#) statement that retrieves the ID and Name of all the staff that has BTech Qualification.

The staff details that are in the staff table has to be retrieved based on the qualification that is in the Qualification table. Hence we have to use [Subquery](#) within the SELECT statement.

```
INSERT INTO HighQual (ProductID, ProductName)
    SELECT ProductID, ProductName
    FROM Staff
    WHERE StaffID IN (SELECT staffID
                      FROM Qualification
                      WHERE QualName='BTech'
                    );
```

Here we used the [IN](#) operator since the inner subquery may return more than one value.

14.1 How do you change the existing data?

We need to change the description of CBSE syllabus as 'Central Syllabus' and ISCE Syllabus as 'Special Syllabus'

Solution:

We change the existing data using the [UPDATE statement](#).

First we change the description of the CBSE syllabus to 'Central Syllabus'

UPDATE Syllabus SET Description = 'Central Syllabus' WHERE SyllabusName = 'CBSE';

Here we identified the row to be updated by filtering it using the [WHERE clause](#).

UPDATE Syllabus SET Description = 'Special Syllabus' WHERE SyllabusName = 'ICSE';

Notice how we have enclosed [character string data in single quotes](#).

14.2 Update the total score column with the total score for each candidate?

The scores of the students in three tests are in the Marks table. But the Total score column does not have data. We need to update the total score column of the Marks table with the sum of the three marks for all the students.

Solution

The solution to this problem is very simple.

To make the changes to existing data we use the UPDATE statement and to make the statement affect all the rows, we omit the WHERE clause.

```
UPDATE Marks SET Total = Mark1 + Mark2 + Mark3;
```

14.3 Change the name of all products by adding 'X' at the end of the name of all products that has price lower than 20.

Solution

To change the data in the existing rows we use the update statement. To concatenate character strings we can use [concatenation operator](#) of the [concat function](#).

```
UPDATE Product
    SET ProductName=ProductName||'X'
    WHERE Price <20;
```

14.4 Change the name of all schools that begin with ABC to XYZ International School in the Qualification table of project management

Solution

To change the existing rows we use the [UPDATE](#) statement. To find data that match a pattern we can use the [LIKE](#) operator and the [wild cards](#).

UPDATE Qualification

SET Institute = 'XYZ International School'

WHERE Institute LIKE 'ABC%';

15.1 How do we remove the data of all students whose name begins with 'A' and Total score is below 200?

Solution

To match patterns like find names that start with 'A' we need to use the [wild card](#) % with the [LIKE](#) operator.

```
DELETE FROM Student WHERE StudentName LIKE 'A%' AND Total <200;
```

Here 'A%' means that the first letter of the name should be 'A' rest of the characters can be anything and any number of characters.

The two conditions are combined together with the logical operator 'AND'

15.2 How do we remove all the students who were born in the year 98?

Solution

To find the students who were born in the year 98 we can use pattern matching using [LIKE](#) operator and the [wild cards](#).

```
DELETE FROM Student WHERE DateofBirth LIKE '%98';
```

This can also be solved using the [conversion function](#) TO_CHAR to retrieve the year part from the date like this.

```
DELETE FROM Student WHERE TO_CHAR (DateofBirth, 'YY') = '98';
```

Here 98 is given in single quotes since the data returned by the TO_CHAR function is in the form of character string.

15.3 How do we remove the data of all the batches that are not following Syllabus S20?

Solution

```
DELETE FROM Syllabus WHERE NOT SyllabusID = S20;
```

Here the logical operator NOT is used to identify the Syllabus that is not S20.

15.4 Remove the skills that have second letter as 'C';

Solution

To remove the data we use the [DELETE](#) statement.

We filter data that match a pattern using wild card '_'

```
DELETE FROM Skills  
WHERE SkillName LIKE '_C%';
```

To indicate that the 'C' can be preceded by a single character which can be anything the '_' is used. I gave a '%' sign at the end to indicate that any characters of any number can come after 'C'.

16.1 How do we retrieve the data in the format “Average mark of *StudentID* is 80” for all the students? The data should come under the heading ‘Names and Percentages’.

Solution

The data that is retrieved from the table Marks are StudentID and Total Mark. Character strings are concatenated to this to give it the form of a sentence.

Here we use the Concatenation operator to concatenate the character strings and the data from the table.

```
SELECT 'Average mark of ' || StudentID || ' is ' || Total/3 "Names and Percentages" FROM Marks;
```

We used the column alias to give a custom heading to the output. We enclosed the column alias within double quotes since there are blank spaces in the column heading.

Instead of the concatenation operator we could also have used the concatenate function.

An important point to be noted: Here we need to remember that instead of the arithmetic calculation Total/3, we CANNOT use the aggregate function Avg() since Avg() function operates on the values of a column, but here we need the Average marks of each candidate which needs operation on each row.

16.2 How do we retrieve the name and grade of all the students who are born in the month of January? The Student Name column should have the heading "Name" and ID should have the heading "Student Code". Add a serial number at the beginning.

Solution

There are two ways of solving this problem.

By matching the pattern or by using the TO_CHAR conversion function. We would see both.

To match the pattern, we use the LIKE operators and the wild cards.

To add serial number like numbering we use the [pseudo column](#) ROWNUM.

```
SELECT ROWNUM "Serial Number", StudentName Name, StudentID "Student Code"
      FROM Student WHERE DateofBirth LIKE '%JAN%';
```

Here the column alias for StudentName is not given in quotes since it is a single word. But if you wanted to see the column in sentence case you could have used quotes. Other wise we would see it in upper case by default. The other way of solving is by using the TO_CHAR function.

```
SELECT ROWNUM "Serial Number", StudentName Name, StudentID "Student Code"
      FROM Student WHERE TO_CHAR (DateofBirth, 'MON') = 'JAN';
```

Important: Note that the name of the month JAN is given in upper case in both the queries since that is how it is stored in the database. If you gave the name of the month in any other case we would not get the answer.

16.3 How do we retrieve the ID of all students who have scores between 400 and 450? Include these two scores in the search.

Solution

We can filter the data using combination of comparison operators \geq , \leq and logical operators OR by using the BETWEEN operator.

```
SELECT StudentID FROM Student WHERE Total >=400 AND Total<=450;
```

We can also use the [BETWEEN operator](#) which has the same effect.

```
SELECT StudentID FROM Student WHERE Total BETWEEN 400 AND 450;
```

16.4 How do we retrieve the all the marks, minimum mark and percentage for each student. Assume that the maximum score for each subject is 100?

Solution

To find the smallest from a set of values we can use the LEAST function and the percentage can be found using the arithmetic operators.

```
SELECT StudentID, Mark1, Mark2, Mark3, LEAST (Mark1, Mark2, Mark3) "Least Score",  
       Mark1, Mark2, Mark3/3 "Percentage Score" FROM Marks;
```

Important: Here the aggregate function MIN () CANNOT be used in the place of least since MIN function acts on the values of a column. But we want to find the minimum values from a set of values of the same row. Hence we used the LEAST function.

16.5 Retrieve all the data in the Category table.

Solution

This is done by a simple SELECT statement.

```
SELECT CategoryID, Name FROM Category;
```

We could have done the same thing using

```
SELECT * FROM Category;
```

But it is always a good practice to explicitly mention the name of the columns in the SELECT clause without relying on the order of the columns to identify the columns.

17.1 How do we retrieve the names and marks of all students in the descending order of marks? For students having the same marks the details should be displayed in the ascending order of name. The first column should be SerialNumber with each row having serial number 1, 2, 3 etc.

Solution

To sort data we use the [ORDER BY](#) clause. For the special effect of students having the same marks the details should be displayed in the ascending order of name we include the StudentName also in the ORDER BY Clause in the Ascending order.

The marks and name are in two different tables, so we would use the [JOIN](#) to retrieve data from two tables which are related using the StudentID. So we would use the [equijoin](#)

The Serial numbers can be given by using the [pseudocolumn ROWNUM](#)

```
SELECT ROWNUM "Sl. No.", StudentName, Total
      FROM Student JOIN Marks USING (StudentID) ORDER BY Total DESC, Name ASC;
```

Here a special heading is given to the [pseudocolumn](#) using [Column Alias](#)

17.3 What is the meaning of the statement `SELECT StudentName FROM Student Order BY 1;`

Solution

Here we have a `SELECT` statement with an `ORDER BY` clause. The [order by clause has a number instead of a column name](#).

`SELECT StudentName FROM Student Order BY 1;`

The meaning of the `SELECT` statement is to retrieve the Studentname in the ascending order of the first column in the `SELECT` clause, which is the `StudentName` column.

17.4 How do we retrieve the student names and date of birth in the ascending order of their age?

Solution

We can solve this by calculating the age and arranging the data in the ascending order of the age. BUT an easier way which is also faster is to arrange the data from the latest birth date to the earliest birth date. This would have the same effect of arranging the data in the descending order of the date of birth. This is because, the person with the latest birth date will be youngest and the person with the earliest birth date would be the oldest.

So we can write the query as

```
SELECT StudentName, DateofBirth
FROM Student
ORDER BY DateofBirth DESC;
```


17.5 To assign to a project we need to find staff with Java and database skills. How do we do this? Sort in ascending order of skills

Solution

This problem may seem straight forward but the solution is more complex than it seems.

Let us analyse it.

Here the name of the staff from the Staff table has to be retrieved using the skill set from the Skill table. So we need a [subquery](#) that returns the right set of staffIDs.

We also need to sort the data according to the skill name. So the simplest solution would be to use a [JOIN](#) also.

In the Skill table, the skills of a staff are represented as separate rows. So to identify the staff with multiple skill sets it is not sufficient that we use a combination of condition check and logical operator like this

```
WHERE SkillName = 'Java' AND 'SkillName='Database';
```

The AND operator can be used if both these data were in the same row. But as we saw the different skills are represented by different rows.

So to identify staff with multiple skill sets we need to use the SET operators.

```
SELECT Name
FROM Staff JOIN Skill
      USING (StaffID)
WHERE StaffID IN (
      SELECT StaffID
      FROM Skill
      WHERE SkillName='Java'
      INTERSECT
      SELECT StaffID
      FROM Skill
      WHERE SkillName='Database')
ORDER BY SkillName;
```

Here we used the INTERSECT to ensure that only the staffIDs that have both the skills are retrieved.

17.6 To assign to a project we need to find staff with at least one of the skills Java or database skills. How do we do this? Sort in descending order of skills

Solution

Here the name of the staff from the Staff table has to be retrieved using the skill set from the Skill table. So we need a [subquery](#) that returns the right set of staffIDs.

We also need to sort the data according to the skill name. So the simplest solution would be to use a [JOIN](#) also.

In the Skill table, the skills of a staff are represented as separate rows. So to identify the staff with multiple skill sets it is not sufficient that we use a combination of condition check and logical operator like this

```
WHERE SkillName = 'Java' AND 'SkillName='Database';
```

The AND operator can be used if both these data were in the same row. But as we saw the different skills are represented by different rows.

So to identify staff with multiple skill sets we need to use the SET operators.

```
SELECT Name
FROM Staff JOIN Skill
      USING (StaffID)
WHERE StaffID IN (
      SELECT StaffID
      FROM Skill
      WHERE SkillName='Java'
      UNION
      SELECT StaffID
      FROM Skill
      WHERE SkillName='Database')
ORDER BY SkillName DESC;
```

Here we used the UNION to ensure that only the staffIDs that have at least one of the skills are retrieved.

18.5 How do we retrieve the Names of all students and NCC rank for only those students who are part of NCC?
The two tables are related to each other using the StudentID with Student ID being the primary key in the Student table and foreign key in the NCC table.

Solution

The data that whether a student is part of NCC is and its related data are in the NCC table. The two tables are related to each other using the StudentID. So to find the students who are in the NCC table use the [Equi Join](#)

```
SELECT Name
      FROM NCC
     JOIN Student
    USING (StudentID);
```

We used the USING clause instead of the ON clause since the columns involved in the Join condition are having the same name.

18.6 How do we retrieve the name, rank and age of all students and NCC rank irrespective of whether the student is part of NCC GROUP?

Solution

The data that whether a student is part of NCC is and its related data are in the NCC table. The two tables are related to each other using the StudentID. We need the names and ages of all the students and the NCC ranks of students who are part of NCC

That is; all rows from the Student table and the corresponding rank from the NCC table for all the students who are also part of NCC. So we use the Outer Join

```
SELECT Name, Rank, MONTHS_BETWEEN(DateofBirth, Sysdate)/12      "Age"
      FROM NCC RIGHT OUTER JOIN Student USING (StudentID);
```

To calculate the age we used the months_between function and divided its result by 12. An alias name "Age" is given to this [virtual column](#).

Here we used the RIGHT OUTER JOIN since the table from which we need all the rows – Student Table – is on the right side of the JOIN keyword.

18.7 How do we retrieve the name of the student and the grade in academics based on the total annual score. Retrieve the data as "StudentName has # % ". Sort the data in the descending order of the score. Include only the details of those students who are in Batch B1 and whose scores are in the range 200 to 250.

Solution

The name of the student is in Student table and the marks are in Marks table. The two tables are related to each other by referential integrity constraint applied to StudentID column. So to retrieve the data from two tables we need to use an Equi Join

To concatenate character string with data we can either use the concat function or the concatenation operator.

```
SELECT Name||" has " || Total/3 ||" %"
       FROM Student JOIN Marks USING (StudentID)
       WHERE BatchID='B1' AND Total BETWEEN 200 AND 250
       ORDER BY Total DESC;
```

We used the ORDER BY to sort the data according to the Total marks. The logical operator AND was used to combine the effect of the two conditions.

18.8 How do we retrieve the names of students and their team leaders? Also include the team leaders themselves who have no team leaders. Sort the data in the ascending order of the student's name.

StudentID	TeamLeaderID	StudentName	DateofBirth	BatchID
ST1		Mathew Haiden	01-Jan-97	B1
ST 2	ST1	Mary Claire	28-Mar-96	B1
ST 3	ST1	Muhammed Ali	15-Jun-95	B1
ST 4	ST5	Elton John	11-Sept-98	B2
ST 5		Hanna Montanna	31-Dec-98	B2

Solution

From the sample data in the table we can see that the relationship of team members and team leaders are set by using the teamLeaderID column.

The team leaders do not have TeamleaderIDs. But the team members do. Here since the relationship exists within the table we use the [SELF JOIN](#) to solve the problem with the JOIN condition

TeamMember's TeamLeader ID = TeamLeader's StudentID.

Since also need the details of the team leaders who do not satisfy the mentioned JOIN condition, we need to use the [Outer Join](#)

```
SELECT Mbr.Name, Ldr.Name
      FROM Student Mbr LEFT OUTER JOIN Student Ldr
      ON (Mbr.TeamLeaderID=Ldr.StudentID)
      ORDER BY Mbr.Name ;
```

Here appropriate table alias are used to identify the team members name and the team leaders name in the SELEC clause. If we do not do this w would get the column ambiguity error.

We used the [ORDER BY](#) clause to sort the data.

18.9 How do we retrieve the names of all the students who are in Standard 1 batch A and following CBSE Syllabus?

Solution

This question seems very simple and straight forward. But, when we analyze it we can find out that the name is in the Student table, the batch name is in the Batch table and the standard in the Standard table and the Syllabus name is in the Syllabus Table.

So in order to retrieve the needed data we need to retrieve it from four tables. Hence we use the mutable join
SELECT Name FROM Syllabus

```
        JOIN Standard
            USING (SyllabusID)
        JOIN Batch
            USING (StandardID)
        JOIN Student
            USING (BatchID)
        WHERE StdName='I' AND BatchName='A' AND SyllabusName='CBSE';
```

Let us add some query optimisation techniques to this.

When we join more than two tables, we need to mention the table with lower number of rows first since it is a [performance Tuning](#) tip that would improve performance.

The query that we wrote is already following this rule.

The second rule is to filter at the earliest possible time. Here the earliest possible filter point is the ON clause. So we can modify the query like this:

```
SELECT Name FROM Syllabus sy
        JOIN Standard Sd
            ON (Sy. SyllabusID = Sd.SyllabusID AND StdName='I' AND SyllabusName='CBSE')
        JOIN Batch b
            ON (b. StandardID = Sd.StandardID)
        JOIN Student St ON (b. BatchID = St.StandardID AND BatchName='A' );
```

18.10 How do we find the name of the products that have reached the reorder level?
Use the retail database for reference.

Solution

Here we need the product name from the product table based on the reorder level data in the Stock table.
Though this involves two tables, we need data only from the product table. So here we need only a subquery.

```
SELECT ProductName
      FROM Product
     WHERE ProductID IN (SELECT ProductID
                        FROM Stock
                       WHERE Quantity <= ReorderLevel);
```


18.11 Product name and Order details of the product 'Mavila'

Solution

The product name is from the Product table and Order details from the Order Table.
Since we need to retrieve data from both tables we use JOINS.

```
SELECT ProductName, OrderDate, Quantity
FROM Product Pr
JOIN Order Or
ON(Or.ProductID = Pr.ProductID AND ProductName='Mavila');
```

19.3 How do you retrieve the nth largest element from?
Find the mark of the person who got the second rank.

Solution

We need to find the second largest score from the Marks table and there are two ways to solve this.
Let us first look at the popular solution.

Method 1:

Here we use the concept of [correlated subqueries](#) and [aggregate functions](#).

The general syntax for the question is

```
SELECT Total from Marks Outer WHERE n-1 =  
        SELECT COUNT (Total) FROM Marks Inner  
        WHERE Inner.Total>Outer.Total;
```

Where n is the any whole number. The Marks table used in the outer query is given the alias 'Outer' and in the inner query it the table alias 'Inner' is used.

So to find the second highest score, we substitute 2 for n.

```
SELECT Total from Marks Outer WHERE 2-1 =  
        SELECT COUNT (Total) FROM Marks Inner  
        WHERE Inner.Total>Outer.Total;
```

Method 2:

In this method we use the concepts of [inline views](#), [pseudo columns](#) and [aggregate functions](#)

The general Syntax would be

```
SELECT Min(Total) FROM  
        (SELECT Total FROM Marks ORDER BY Total DESC)  
        WHERE ROWNUM<n;
```

So to find the second highest score, we substitute 2 for n.

```
SELECT Min(Total) FROM  
        (SELECT Total FROM Marks ORDER BY Total DESC)  
        WHERE ROWNUM<2;
```

19. 4 How we retrieve the details of all the students who are in having the same total score as 'Tony George'? Do not include details of 'Tony George' in the result

Solution

Here we need to retrieve the data based on the person 'Tony George'. So we need to use subqueries. The name and total are from two different tables so we need to use JOIN also

First, find the total marks of the student named Tony George – in the inner query.

Second Find the details of students who have the same total as Tony George – in the outer query

```
SELECT Name, DateofBirth, Total
      FROM Student JOIN Mark USING (StudentID)
     WHERE Total=
           (SELECT Total FROM Student JOIN Marks USING (StudentID)
            WHERE Name='Tony George');
```

19.5 How do we retrieve the name of the batch that has the maximum number of students?

Solution

Here we need to use a series of [subqueries](#).

First to find the maximum strength;

Second to find the batch id of the batch with the maximum strength and

Third, to find the name of the batch with this batchID.

```
SELECT BatchID, BatchName FROM Batch
      WHERE BatchID =
            (SELECT BatchID FROM Student
             GROUP BY BatchID
             HAVING Count (BatchID) =
                  (SELECT Max (Count (BatchID)) GROUP BY BatchID));
```

19. 6 How do we retrieve batchname, number of students for each batch? Sort the data according to the batch name.

Solution

To find the batch name and the strength of each batch we can use a special [subquery](#) - subquery in the SELECT clause.

```
SELECT BatchName, (SELECT Count (studentID) FROM Student WHERE batchid = Outr.batchid)
                  FROM Batch Outr ORDER BY BatchName;
```

Here the inner query in the SELECT would find the strength for batched of each batched retrieved by the outer query.

Finally the data is sorted by batchname using the ORDER BY clause.

19. 7 How do we retrieve details of the students who earn first three positions for Batch B1?

Solution

To solve this we need to use a combination of [Inline views](#) and [pseudo columns](#).

```
SELECT * FROM
    (SELECT Name, Total FROM Student
      JOIN Marks USING (StudentID)
      ORDER BY Total DESC)
WHERE ROWNUM < 4;
```

You may wonder why the query cannot be written like this without using an inline view:

```
SELECT Name, Total
FROM Student JOIN Marks USING (StudentID)
WHERE ROWNUM < 4
ORDER BY Total DESC;
```

This query **Will not** give the desired results since according to the [order in which the query is parsed](#), the WHERE clause works before the ORDER BY Clause.

So the parser would first filter and retrieve the first four rows and then sort it in the descending order of the Total.

19. 8 How do we display the scores of all the team leaders?

Solution

If we analyze the question we can see that details on who is the team leader are in the Student table and scores are in the Marks table. So we use a [JOIN](#)

In addition to this, the team leaders can be identified by retrieving the rows that has NULL in the ManagerID column

So the query is:

```
SELECT StudentName, Total
      FROM Student JOIN Marks USING (StudentID)
     WHERE ManagerID IS NULL;
```

19.9 How do we display only the odd numbered records?

Solution

This question is more of a brain exercise and it can be solved using [Inline views](#) and [pseudo columns](#). Let us use the Student table.

```
SELECT * FROM  
    (SELECT ROWNUM Rnum, StudentID, StudentName FROM Student) Inner  
WHERE MOD (Inner.Rnum, 2) <>0;
```

Here the odd rows are identified by the simple arithmetic logic of checking the modulus of the row number and 2. When the modulus is not zero, it is an odd row.

19.10 Retrieve the work sheet including the type of work of Tony from the project management database. Use the Project Management database.

Solution

Since we need WorkSheet and the Workcategory we use the JOIN. To filter the data based on the name of a Staff we use the subquery.

```
SELECT WorkDateTime, Name "Work Category", DurationMinutes
      FROM WorkCategory
      JOIN WorkSheet
      ON (WcatID = WorkCatID AND StaffID IN (SELECT StaffID
                                           FROM Staff
                                           WHERE Name='Tony'))
      ;
```

20.1 How do we find the age of all the students who were born after 1989?

Solution

To calculate the age we can use a combination arithmetic operation and date function Months_between()

```
SELECT Months_Between(Sysdate, DateofBirth)/ 12 "Age"  
FROM Student  
WHERE DateofBirth > To_Date('31-DEC-1989', 'DD-MON-YYYY');
```

Here, to compare the date of birth with the last day of 1989, we need to use the To_Date function which is one of the [conversion functions](#).

So even though WHERE DateofBirth > '31-DEC-1989' looks logical, it would not parse.

This is because '31-DEC-1989' as such will be parsed as character string and not as a date. To convert this character string to date in the default format, we need to use the To_Date function.

20.2 How do we find the batch name and the number of students in each batch?

Solution

To solve this problem, we need to first group the data on the basis of the Batch and then find the strength of each batch. This can be found from the Student table.

To group the data we use the [GROUP BY](#) clause and to compute the strength of each group we use the [aggregate function Count\(\)](#)

The batch name is in the batch table. So to solve the whole problem we need to retrieve data from two tables and hence we use a JOIN.

```
SELECT BatchName, Count (StudentID)
      FROM Batch JOIN Student USING (BatchID)
      GROUP BY BatchName;
```

Here there is a specific reason why the BatchName is used to group the data rather than the batched. This is because the batch name should also be included in the SELECT clause and the rule for the GROUP BY clause is that, only the column name used in the SELECT clause as such.

Here the Batch table is mentioned first in the JOIN since it is a [performance tuning](#) tip to mention the table with the lower number of rows first in the JOIN.

20.3 How do we display the names in Sentence case and total marks of all candidates in the format 1,234.89? Assume that the column, 'Total' can hold decimal numbers upto a precision of two decimal places. (Number(6,2))

Solution

The case of characters is changed to sentence case using the [character function Initcap\(\)](#) for case manipulation.

To display a number in any specific format, we can use the [conversion function To_Char\(\)](#).

Since the Total and the name are in two different tables, Marks and Student we need to use a Join. These two tables are related using referential integrity. So we can use [Equi join](#) to retrieve the corresponding rows.

```
SELECT Initcap(StudentName), TO_Char(Total, '9,999.99')
      FROM Student JOIN Marks
      USING (StudentID);
```

20. 4 How do we display the date of birth of all the candidates in the format “first of June, 1999”? The data of birth should come under the heading ‘DateofBirth’.

Solution

To display a date in any desired format other than the default format, we need to use the conversion function [To_Char\(\)](#).

We specify the desired format as the second argument of the To_Char function.

```
SELECT To_Char(DateofBlrth, 'DDspth "of" Month, YYYY') DateofBirth FROM Student;
```

In the format, note that the character string ‘of’ is given within double quotes. Any character string other than symbols like comma, semicolon, colon etc. should be given in double quotes.

Note that we have given a column alias to the column. But why did we do it if we needed the column name itself to come as the heading?

It is because, if we do not give the column name, the heading that would come would be ‘To_Char (DateofBlrth, ‘DDspth “of” Month, YYYY)’. This is a virtual column since it is a derived column and it does not correspond to any column in the database.

20.5 How do we display the date of birth of all the candidates in the format “June One, Nineteen Ninety Nine 10:25 pm”?

Solution

To display a date in any desired format other than the default format, we need to use the conversion function [To_Char\(\)](#).

We specify the desired format as the second argument of the To_Char function.

```
SELECT To_Char(DateofBirth, 'Month DDsp, Year HH:MI am') DateofBirth FROM Student;
```

The Hour and minute is displayed using the format HH and MI. To display the ‘am’ or ‘pm’ depending on the time, we just need to specify ‘am’ in the format. The database will interpret the time and display ‘am’ or ‘pm’.

20.6 How do we display the date of the next Monday that fall after today in the format 'Monday, August 30, 98'?

Solution

To find the date of the next Monday that comes after today, we use the [date function](#) Next_day().

To display a date in any desired format other than the default format, we need to use the conversion function [To_Char\(\)](#).

We specify the desired format as the second argument of the To_Char function.

Since both functions are to act on the same data, we nest the Next_Day function within the To_char function.

```
SELECT To_Char(  
           Next_Day(DateofBirth, 'Monday')  
           , 'Day, Month DD, YY') DateofBirth  
FROM Student;
```

20.7 How do we display the batch name, name short forms and age of all the students in the ascending order of their age? Display only the details of students who are older than 12.

To make the short forms of the names remove any occurrences of 'a' or 'e' from the names.

Solution

To find the age we can use a combination of arithmetic operations and the months between function.

To remove any occurrence of the 'a' or 'e' from the name we can use the [replace function](#).

The name of students and batch names are in different tables. So we use [JOINS](#).

```
SELECT BatchName, Replace (Replace (StudentName,'a'), 'e'),  
       Months_Between(Sysdate, DataofBirth)/12 "Age"  
       WHERE Months_Between(Sysdate, DataofBirth)/12 >12  
       ORDER BY Age;
```

Here we notice that the Column alias 'Age' is used in the ORDER BY clause. But, though the WHERE clause also uses the age for comparison with 12, it does not use the Column Alias. Instead we chose to write the calculation of the age all over again.

Why is this?

From this query we can learn one important rule in the [Order of query parsing](#). In this query, the WHERE-filter is parsed first, then the [Column Alias](#) is given and finally the [ORDER BY](#) Clause works to sort the data.

So when the WHERE clause works; the column alias is not yet assigned and so the age calculation has to be given explicitly. But by the time the ORDER BY clause works the column alias is already assigned and hence it can use the column alias 'Age' instead of the calculation.

20.8 How do we display pass for all students with marks 100 and 149 distinction for marks 250 and above and failed for marks below 100? Display appropriate heading for the output. Include the name of the student also in the output.

Solution

Here I choose to display the name and the status of each candidate. We need the output according to the conditions given. So we can choose to use the CASE statement.

```
SELECT StudentName, Total, (CASE
                                WHEN Total >=250 THEN 'Distinction'
                                WHEN Total BETWEEN 100 and 249 THEN 'Pass'
                                WHEN Total<100 THEN 'Fail'
                                END
                            ) Status
FROM Student JOIN Marks
USING (StudentID);
```

Here 'status' is the column alias given to the output of the case statement.

20.9 How do we find a unique code for all the students by concatenating the student ID, first three letters of the name in upper case, their batch name and standard name?

Solution

To retrieve substring from the main string we use the character manipulation [function Substr\(\)](#).

Use either the [Concat function](#) or the concatenation operator for concatenating the strings.

Use the case manipulation function [Upper function](#) to change the case to upper case.

The details of the student, the batchname and standard names are in three different tables. So we use the [mutable join](#).

So to solve this we use some [character functions](#) and mutable join.

The query is:

```
SELECT StudentID || Upper( Substr(StudentName, 1,3)) || BatchName
      FROM Standard JOIN Batch
            USING (StandardID)
      JOIN Student
            USING (BatchID);
```

20.10 How do we find the name of the student their marks for the different subjects and the highest score from among their scores for different subjects? Sort the result in the descending order of the highest score. For students having same highest scores sort them in the ascending order of the names.

Solution

Student name and marks are in different tables, so we use the [JOIN](#).

```
SELECT StudentName, Mark1, Mark2, Mark3, Greatest (Mark1, Mark2, Mark3) "Highest score"  
      FROM Student JOIN Marks  
      USING (StudentID)  
      ORDER BY "Highest score" DESC, StudentName;
```

We gave a meaningful column heading to the [virtual column](#), highest score. This column alias is also used in the [ORDER BY](#) clause.

Important: We cannot use the aggregate function Max in the place of Greatest, since Max works on the values of a column and finds the largest, but here we need to find the largest from a set of values in a row.

We were able to use the column alias 'Highest score' in the ORDER BY clause, since if we see the [order of SQL Query parsing](#), the ORDER BY clause is parsed after the column alias is assigned.

20.11 How do we create a list of company name short forms? Short forms are created by taking the first three letters of the company name.

Use the retail shop database.

Solution

This problem can be solved by a simple [SELECT](#) statement which uses the [SubStr](#) function

```
SELECT SubStr(CompanyName, 1,3) "Short Form" FROM Company
```

20.12 How do we retrieve the prices of all the tooth care products rounded to the nearest 10th place? Give the column heading as "Approximate price". Sort it in the ascending order of the price. Use the Retail Store database.

Solution

To round the data to the nearest tenth place we use the [ROUND](#) function.

```
SELECT Round(Price, -1) "Approximate price"  
      FROM Product  
      ORDER BY "Approximate price" ;
```

We used the negative argument, -1, in the Round function to give it the special behaviour of rounding to the nearest tenth place.

We used the [column alias](#) in double quotes since it has a space in between.

We use the column alias in the ORDER BY clause since we can take advantage of the [order in which the query is parsed](#).

20.13 How do we retrieve the price of all products costlier than 1000 truncated to the nearest 100? The output should be in the format \$1,200.00 under the heading "Price after Discount"
Use the retail store database.

Solution

To truncate the price use the Trunc function and to format the number use the To_Char function with desired number format.

```
SELECT To_Char(Trunc(Price, -2), '$9,999.00') "Price after Discount"  
      FROM Product  
      WHERE Price>1000;
```

We used the negative argument, -2, in the Round function to give it the special behaviour of rounding to the nearest hundredth place.

We used the [column alias](#) in double quotes since it has a space in between.

20.14 The retail store wants to develop a code list for the company names by replacing 'C' with 'D'; 'A' with 'B' and 'P' with 'Q'. How do we do this?

Solution

To replace a set of characters with corresponding characters in another set as mention in the question, we use the [translate](#) function.

```
SELECT Translate (ProductName, 'CAP', 'DPQ') Codes FROM Product;
```

20.15 How do we retrieve the category name, Company name, product name and the price?
Use the Retail store database

Solution

These data are from the tables Company, Category and Product. Since there are more than two tables, the mutable join is used.

```
SELECT Cm.Name "Company Name", CatName "Category Name", Pr.Name "Product Name", Price
FROM Company
      JOIN Category
            ON(CompanyCode = CompanyID)
      JOIN Product
            USING(CategoryID);
```

As a [performance tuning](#) step we mentioned the tables in the Joins in the ascending order of the number of rows in the table.

It is always a good point to use [column alias](#) whenever we use a table alias to differentiate the column name.

20.16 How do we retrieve the name and salary of the first three senior most staff in the company?
Use the project management database.

Solution

We can find the first n senior most staff using a combination of the [pseudocolumn ROWNUM](#) and the [Inline views](#).

```
SELECT Name, Salary
       FROM (SELECT Name, Salary
              FROM Staff ORDER BY Salary DESC)
      WHERE ROWNUM <4;
```

The inline view would give the data in the descending order of salary and the outer query would choose the first three rows from it.

We used ROWNUM <4 since we need the first three highest earning staff.

Here we have to use the inline views since the query if written without inline views as **the one below would not give the desired results**. This is because if we look at the order in which query is parsed, it would take the first four rows retrieved and then Sort it in the descending order. But this is not what we need.

```
SELECT Name, Salary
       FROM Staff
      WHERE ROWNUM <4
     ORDER BY Salary DESC;
```

20.17 How many projects has the company done in the Finance domain?
Use the Project management database.

Solution

Here we retrieve the count of projects from the Project table based on the filtering done according to the domain name which is in the ProjectDomain table.

So here we do Not need a Join. Here we use subquery.

```
SELECT count (projectid) "No: of Finance Projects"  
      FROM Project  
     WHERE domainID=  
           (SELECT DomainID  
            FROM ProjectDomain  
            WHERE Name='Finance');
```

A meaningful [column alias](#) is given for the count retrieved.

20.18 Domains along with the projectnames that have projects longer than 2 months

Solution

We retrieve data from two tables Project and ProjectDomain based on the filter done using data from the Project table.

So we use [JOIN](#) and [subquery](#). To find the number of months between the startdate and the end date of the project we can use the [date function](#) months_between()

```
SELECT Dom.Name, Pr.Name
      FROM ProjectDomain Dom
     JOIN Project Pr
        ON (Dom.ProjectID= Pr.ProjectID AND DomainID=
            (SELECT DomainID
             FROM Project
             WHERE Months_Between(EndDate, StartDate)>2
            ));
```

Here we added the filter to find the projects longer than 2 months in the ON condition as a query optimisation technique to [improve the performance](#). It is always good to filter at the earliest point.

21.4 How do we find the batch name and maximum score in each class?

Solution

To find the maximum score in each batch we need to group the data based on batch and then find the highest score from each batch.

We need batch name from the batch table and the score from the Marks table. But there is no direct relationship between the batch table and the Marks table. The relationship between the two is through the Student table. So we need a JOIN three tables and not two.

```
SELECT BatchName, Max (Total) "Highest Score"  
      FROM Batch JOIN Student  
                USING (BatchID)  
      JOIN Marks  
                USING (StudentID)  
      GROUP BY BatchName;
```

We have also given a meaningful column alias name for the maximum score in each batch.

We used the BatchName in the GROUP BY clause since we also needed to use it in the SELECT clause. If we had not done this, we would need to use an [inline view](#) to solve the query.

21.5 How do we retrieve the number of students in each batch? Sort the data in the descending order of the strength.

Solution

To find the number of student in each batch we need to group the students based on the batch and then find the number of students in each group.

To group the data we use the [GROUP BY](#) clause and to find the number of rows in each group we use the aggregate [function Count](#) ().

```
SELECT BatchID, Count (StudentID) "Strength"  
      FROM Student  
      GROUP BY BatchID  
      ORDER BY "Strength";
```

We are able to use the column alias in the ORDER BY clause by taking advantage of the [order in which the query](#) is parsed.

21.6 How do we retrieve the date of births and the number of students sharing the same date of births? Sort the data according to the dates of birth.

Solution

First we group the students based on their dates of birth and then find the number of students in each group. To group the data we use the [GROUP BY](#) clause and to find the number of rows in each group we use the [Count function](#).

```
SELECT DateofBirth, Count (StudentID) "Numbers "  
FROM Student  
GROUP BY DateofBirth  
ORDER BY DateofBirth;
```

21.7 How do we find the total quantity of the different products (along with the product names) that has been ordered so far, with the date on which the last order was made? Display the data in the descending order of the quantity.

Use the retail database.

Solution

To solve this we use the Order table since it has the order date and the data regarding all the orders placed for all the products.

To find the total quantity and the latest order data, first group the products according to the productID and then take the sum of the quantities of each group. For this we would use the [GROUP BY](#) and the [aggregate function Sum](#) (). To find the latest order date, we use the [Max](#) function.

We need the product name also. So we make the earlier query an inline view from which the total quantity and latest order date from it and [JOIN](#) it with the Product table based on the ProductID. This is because we cannot add the product name in the query that uses the ProductID to group the data.

```
SELECT Name, T.Total "Total Quantity", T.latest "latest date"
FROM Product P
JOIN (SELECT Sum (Quantity) "Total", Max (OrderDate) "latest"
      FROM Order
      GROUP BY ProductID) T
ON (P.ProductID=T.ProductID)
ORDER BY "Total Quantity";
```

This query can also be solved using [GROUP BY](#) and [JOIN](#), like this:

```
SELECT Name, Sum (Quantity) "Total Quantity", Max (OrderDate) "latest date"
FROM Product P JOIN Order O USING (ProductID)
GROUP BY Name
ORDER BY "Total Quantity";
```

Problem

The following are the two tables represented in the [Relational Model](#) with sample data in it. It represents two of the tables of the retail store database.

Category Table

CategoryID	CategoryName	Description
C10	Tooth Care	Tooth care product
C20	Skin Care	Skin care product

Product Table

ProductID	ProductName	Price	Category ID
P10	CP	45	C10
P20	PDT	48	C10
P30	VVL	10	C20
P40	PML	10	C20
P50	PRS	15	C20

Information Given:

- The CategoryID is the [primary key](#) of the Category table.
- The ProductID is the primary key if the Product table
- The categoryID in the Product table is the [foreign key](#) referring to the categoryID (Primary Key) of the Category Table.
- [Database](#) used is Oracle 9i

The report needed is given below

Product	Price	Category	Description
PDT	48	Tooth Care	Tooth care product
CP	45	Tooth Care	Tooth care product
PRS	15	Skin Care	Skin care product
VVL	10	Skin Care	Skin care product
PML	10	Skin Care	Skin care product

The programmer needs to create a query to retrieve the data in the above form. Now how do we retrieve the [data that is stored](#) in the tables in the above format?

The solution

Here the part of the data – product name and the price are stored in the Product table and part of the data – Category name and the category description is stored in the Category table.

So it is clear that we would not be able to use a simple SELECT statement to retrieve the data, since simple SELECT can retrieve data only from a single table.

The solution to this problem is to use a [Join](#).

Here we need the product name, some product details (price) and the corresponding category details. So we need to use the Equi Join to solve the problem.

The column names given in the report are different from the column name in the table. So we need to use the column alias.

The data in the report are sorted in the descending order of the Price and for products having same price the data is sorted in the descending order of ProductName. For sorting we need to use the [ORDER BY clause](#)

So the final solution would look like this.

```
SELECT ProductName AS Product, Price, CategoryName AS Category, Description
FROM Category c JOIN Product p
ON (c.CategoryID = p.CategoryID)
ORDER BY Price DESC, ProductName DESC;
```

Database

Before we go into [what is a database](#), let us understand the background that led to the need of having software. This would help us appreciate the database more.

Consider the scenario of a small shop. What all information do they have to store? They need to store data like Products, its price and stock, the sales and billing details, Accounts details etc

How do they typically store these data? We used to store the data mostly in the register books. They would have a register for Products and prices, Stock, Billing, Accounts registers like the ledger and the journal etc.

Why did they have different registers? Why not store all these info in one register. The different registers are for the following advantages

1. To organize the related data together.
2. When a person wants to know the price of a products, he/ she will know which file to find the data

What practical difficulties will the person managing these data face? They would face the difficulties:

1. To change the existing data. He may have to rub/ strike off the first entry and then add the correction.
2. To remove the unnecessary data
3. Search for a needed info. Though index would help him to some extent, searching for a data in a register may not be very easy
4. Managing the data sometimes become cumbersome.
5. Repetition of the same data in different files becomes necessary for ease

In this scenario where the data is important for the stakeholder, and managing the data in paper registers was not very easy and user friendly, the stage was set for soft wares that managed data for you. This is true especially when the data to be manages was huge.

The first softwares that managed data used file systems. This made the work easy for the person managing the data.

1. He had an easy way to manipulate data
2. Easy way to filter data
3. Easy way for searching data

But the programmer responsible for the creating and maintaining the software faced the following problems.

1. Change the structure of the data to be stored
2. Change the filter conditions
3. Any change in the structure, type or the logic of data storage would mean that the application needed to be changed. Ie; the storage file was dependant on the application. Any change in the structure, type or the logic of data storage meant change in the application also.
4. When the same data had to be shared and manipulated by multiple users simultaneously, it caused problems.

These disadvantages of the file system led to the development of databases, which had the following advantages

1. The software managing the data was independent of the storage system
2. There was a simple language to manage the logical structures for holding the data
3. The same language can be used to manipulate, filter and search the data
4. The data is stored in a server where it can be accessed by multiple users and applications.
5. Database allowed different levels of abstraction

With this background let us summarize what a database is:

The database is an organized collection of [interrelated data](#) stored together without harmful or unnecessary redundancy.

The database helps us to:

1. Manipulate data easily
2. Organize the data in an efficient way
3. Retrieve the data in any needed format
4. Create and change the database structure in the needed format.

5. Interact with the data in the database without worrying about the 'How is it done' part. We need to tell the database only what is to be done. The database will take care of how to do it.

The Database Management System (DBMS) is the software that acts as the interface between the user/application and the database. It is the DBMS that translates the English like SQL to the language that the database would understand.

RDBMS is the [Relational database](#) management system. That is the software that is the interface between the application and a Relational database.

ORDBMS is [Object Relational Database](#) Management System. That is the software that is the interface between the application and the Object Relational Database

The DBMS is an interface between the data and the applications that use the data.

The advantages of using a DBMS are:

Minimal Data Redundancy

The data resides in one central database server and the various programs in the application can access data in different data files. So data present in one file need not be duplicated in another which in turn reduces data redundancy. But there could be reasons for having some amount of redundancy like business rules or performance.

Data stored is Consistent

The DBMS ensures consistency of data even when the data are distributed in multiple tables.

Data Integrity rules can be applied with ease

The functions of the database can be used to enforce integrity rules with ease. The application does not have to code these rules as part of the programming.

Data can be Shared

Since the data is stored in a centralised server, different programmes and applications can operate against the data.

Ease of Application development

With a DBMS adding additional structures and functions are easy. The application programmer does not have to worry about issues like security, data integrity, how the data is stored, concurrent access etc. It is handled by the DBMS. The application programmer needs to implement only the business rules of the application.

Different levels of Data Independence

The different levels of data independence- Physical level, Logical level - are achieved by using DBMS.

Reduces maintenance

Maintenance is reduced since the DBMS handles many of the issues which used to be handled by the application developer. The fact that the database is in a central server also adds to the ease of maintenance.

Functions of a DBMS

Data Definition: The DBMS provides functions to define the structure of the data. These include defining and modifying the table structure, the type and size of fields and the various constraints to be satisfied by the data in each field.

Data Manipulation: DBMS has functions to insert, remove, modify and retrieve the data in the database.

Data Security & Integrity: DBMS contains functions which handle the security and integrity of data in the application. These can be easily invoked by the application and so the application programmer need not code these functions in the programs of the applications.

Data Recovery & Concurrency: Recovery of data after a system failure and concurrent access of records by multiple users are also handled by the DBMS.

Data Dictionary Maintenance: Maintaining the Data Dictionary which contains the meta data of all the objects in the database is the function of a DBMS.

Performance: Optimizing the performance of the queries is one of the important functions of a DBMS. The Query Optimizer is the part of the DBMS which evaluates the different implementations of a query and chooses the best among them.

Characteristics of a good database:

A good database would have the following characteristics.

1. **Performance:** A good database would have the facility for the retrieval and manipulation of data irrespective of the number of tables ensuring the minimum processing time.
2. **Minimal redundancy:** The database system should support minimal redundancy of data. Since redundancy of data would mean that we are using up the valuable disk space.
3. **MultiUser:** A good database should provide multiuser support. This means that multiple users should be able to access the data simultaneously.
4. **Integrity:** When multiple users use the database the data items and the associations between the data should not be destroyed. The integrity of the data should not be lost just because multiple users are using it.
5. **Privacy and security:** The data in the database should be protected against accidental or intentional access by unauthorized persons. The data in the database is valuable and important to the stakeholders. So only authorized users and application should be able to access it.
6. **Database Language:** The database language used should be easy to use and learn and also powerful. The language that is used to communicate with the relational database is SQL. Currently we have the ANSI SQL, which enforces a syntax that can be used in any database. This syntax has been implemented by all the databases to some extent.

Different Levels of Data Abstraction (The three schema architecture of the DBMS)

The different user of the database would need different types of data and at different depths. To enable this, database provides data at different levels of abstraction. That is some users may need some part of the data but may not be interested in how the data is stored in the stored. So the RDBMS hides these data from the users.

This is also described as the three schema architecture of the DBMS. Here schema is the description of the database.

The different levels of data abstraction are

[Physical level](#) (Internal Schema)

[Logical Level](#) (Conceptual Schema)

[View Level](#) (External Schema)

Physical Level or the internal schema is the lowest level of data abstraction. It describes how the data is physically stored in the disk, what are the data structures used for this etc. The internal schema uses physical data model, which describes the complete details of data storage, access paths for the database, and how the data's are retrieved or inserted in the database. Normal users would not need to know this.

The next level of data abstraction is **Logical level** or **conceptual level** which describes what data the database stores and the relationship between the data. The conceptual level has a conceptual schema that describes the whole database for different users who access the database. The conceptual schema hides the details of the physical storage structures and concentrates on entities, relationships, and constraints. The implementation of the relatively simple logical structures may involve complex physical structures. The conceptual model represents the view of data that is negotiated between end users and database administrators

The highest level of data abstraction is **view level** or **External level**. Even though the logical level is simple it may have a variety of information when it comes to large databases. Some users may not need all the information. So the view level abstraction is provided to give them only the needed view of the existing information. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from other user groups. Using the view level abstraction we can have different views of the same data.

How does the three schema architecture lead to efficient computing?

The traditional view focussed on defining data from two different views. The user view and the computer view. The user view, which is similar to the external view, represents data in the form of reports that is needed

by individual users to do their task. The computer view which is similar to the internal view is defined in terms of storage files. This depends upon the computer technology used. So the internal view defined for the initial application could not be used for the subsequent versions of the applications. This led to redundant and inconsistent definition of the same data.

The need for more flexibility in the definition and operation on data, the DBMS was introduced. This increased the performance and efficiency. But since there were only two levels of data representation, there were still problems of redundancy and inconsistency.

This led the ANSI/X3/SPARC study group on DBMS to conclude that for efficient management of data, a third view is needed. This view which is called conceptual view is unbiased towards any single applications data or the physical storage.

So the conceptual view provides a consistent definition of the data and the interrelationship with the data. So the conceptual schema provides views from which any external schema that is needed can be defined.

The introduction of the third view also led to [data independence](#). Any change made to the physical level did not affect the logical level and hence the existing external views were not disturbed. Similarly any change in the logical structure did not affect the existing external schema. This is a big advantage since in the traditional approach any change in any view would lead to the corresponding change in the other views which meant diversion of useful resources.

So with the introduction of the of the three schema architecture there were the following advantages:

- Reduction in redundancy since the whole database could be represented in a single conceptual schema from which any external view can be derived according to the need.
- More consistency of data. The conceptual level ensured that any external view can be derived from the conceptual schema. So there was no need to have any inconsistent definition of data.
- -The application developer could concentrate on the application logic or the business logic, since the problem of data independence was taken care of by the DBMS.

We saw that the introduction of the third view made the DBMS more flexible and adaptable, thus leading to efficient computing.

Data Independence

Why did we need this?

When we used file systems we encountered a problem that any change in the structure of the data represented in the file required a corresponding change in the application. This is because the details of the format of data, the location and how the data is accessed are in the application code itself. This also meant that the application programmers should spent extra time in program maintenance which also meant diversion of resources.

With the introduction of DBMS the programmer was freed from these worries since DBMS allowed data independence. That is in simple terms, changes in one layer did not affect the other existing layers above it.

Data Independence is the ability of the database to represent data without the representation details like location, format etc.

There are two levels of data independence.

First Level (Logical Independence): Any changes in the logical schema do not affect the existing external schema.

Second Level (Physical Independence): Any changes in the internal schema, like physical storage or access methods, would not have any impact on the logical schema.

The schemas mentioned here are the different layers of the database implementation. It should not be confused with the terminology used in the [data models](#). The conceptual schema also called the logical schema constitutes the different tables in the database design.

Three stages of database design:

When we design a database for an application, it is done in different stages.

- First the various requirements are combined to form a **Conceptual Model**.
 - o Data analysis of the design
 - o ER modelling and normalisation
 - o Verification of the design

- The conceptual model is converted into a **Logical Model** compatible with the chosen database management system.
 - o Implement the logical model
 - o Validate the logical model using the principles of normalisation
 - o Assign and validate integrity constraints
 - o Verify the logical model
- This logical model will be internally implemented in the DBMS using the various **Physical Models**.
 - o Estimate storage requirements
 - o Define database security
- The subsets of the logical model that are presented to the user are called the **External Models**.

[What are Data Models?](#)

To understand the concept of data models let us take the example of the register books used in a store in the earlier days. How did a person represent the data in the registers?

The details of the list of products were represented in a tabular format in the product register with columns like product name, price etc.

Product ID	Name	Price	Weight gms
11290	Colgate Active Salt	45	200
11291	Closeup Cool crystals	48	200
11292	Pepsodent	40	200
11293	Meswak	30	200
11294	Mavila	35	200
11295	Lays	10	100

Whenever we have to represent pure data, we need the help of a tool to represent the data. See how the person represented the data regarding the product pricing in the register, He used a table to represent the data. So the model used by him to represent data was a table.

In database we use different data models to represent the data at the different stages of the database design

Data models are a collection of tools for describing data and their interrelationship.

There are different types of data models. They are:

1. Object based logical model
2. Record based logical model
3. Physical data models

Some well known data models that come under the object based logical model are: ER Model and the Object Oriented Model.

[ER model](#)

The ER Model is a data model that is used during the design stage of the database. This model helps the database designer to represent the different entities identified from the problem statement using standard notations. The ER diagram consists of a collection of objects called the entities and relationships among these objects.

Object oriented model

The Object Oriented model is also used during the design phase of a database. This is used to denote the details of the attributes and functionalities in an object that is identified from the problem statement. It is a collection of objects. The objects contain values stored in instance variables. It also has methods (pieces of code) operating on the object.

Record based logical model

In the record based logical model, databases are structured in the form of records of several types and hence called the record based logical model. The different databases we have today are implemented based on the record based Logical models; that is in the databases that are implemented following these models, the data will be represented in the form of records. The different models under this group are the Relational model, Network model and the hierarchical model.

[Relational Model](#)

The relational model uses the concept of a two dimensional table, ie; rectangular structure with rows and columns to represent the data and the relationship between the data.

Hierarchical model

This is one of the first models to be widely accepted and used. In the hierarchical model, the data is represented in tree form with one record superior to another. That is the department branch record is

superior to employee records. The employee records are represented as sub-records of each branch record. In the hierarchical database, the records are represented in a tree structure. So there is a restriction here that a child segment can have only one parent segment. The hierarchical databases are fast, but are inflexible since the relationships are restricted to one to many.

The disadvantage of the hierarchical model is that it is unable to handle many to many correspondences. So it makes the DML operations difficult. It is rarely used today.

Network Model

The Network model is similar to hierarchical model except that a record may have more than one immediate superior. The network model is based on the mathematical set theory. So here the child record can belong to more than one parent. The network model thus allows many to many correspondences.

The hierarchical model uses pointers to relate records where as relational model uses values in the records for relating records

ER Model

The ER Model is the Entity Relationship Model. It is the data model in which the entities to be implemented in the database and the relationship between them are represented using standard diagrammatic representation.

But why should we use this, when describing the design in plain text would serve the purpose?

The standard notations ensure that all the people who go through the ER diagram would understand the same thing that the database designer intended to convey without scope for any confusion. This is because standard notations are used to represent the entities and the relationship between the entities. For the same reason the design represented with ER Model will be without clutter, with only the relevant details mentioned. This is useful especially in the discussions with the top management where ambiguity in the representation of data and its relationship is not acceptable.

The ER model comes under the Conceptual Level of the [different levels of data abstraction](#).

Different types of notations can be used to develop the ER Diagrams. The popular ones among them are:

1. Bachman Notation developed by Dr. Charles William Bachman
2. Peter Chen Notation developed by Dr. Peter Pin-Shan Chen

Both of whom are American computer scientists.

To understand how to use the ER diagram, let us come to the example of the store. Suppose you are given the task of making an application to automate the activities in the store. What are the data that the application needs to maintain?

List the main heads of information that need to be maintained

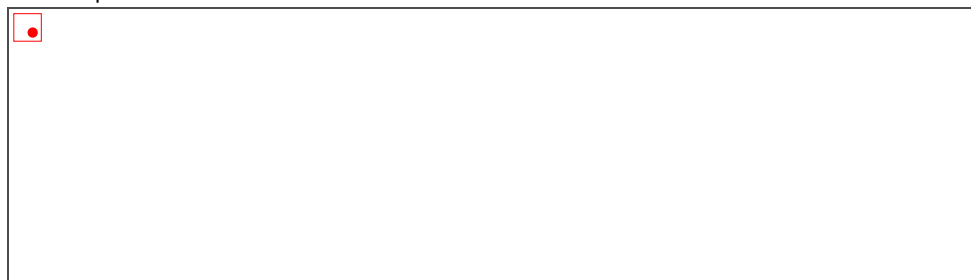
1. Company List
2. Products price
3. Stock
4. Billing
5. Accounts

These main heads are the main “Entities” of the data. The main entities will have subsidiary attributes. We will try to identify the attributes for the first three entities

1. Company – Code, Name
2. Product – Product ID, Name, Price, Weight, CompanyCode
3. Stock – Product ID, Quantity, Reorder level

We would first see the ER model using Bachman notation then go for a comparative study between the two notations.

The ER Model representation



Symbols used for fields:

- # - symbol is used to denote a field used to uniquely identify an item in the entity
- * - symbol is used to denote a field for which there can be multiple instances. Eg: there can be multiple products
- o – Denotes an optional field. That is it is not mandatory, we can leave it blank.

The lines connecting the entities represent the [relationship between the entities](#).

- A single straight line represents the “one to one” relationship
- A straight line with a forked end – also called crow feet - like in the one drawn between Company

and Product entities represents the “one to many” relationship. Here it means that one company can supply many products

- The ‘o’ on the relationship line represents an optional relationship, that is there can be zero to one instances of stock in the Stock table for a Product.
- The ‘|’ on the relationship line represents the mandatory relationship. In the example given we can decipher that a Product should have an associated Company and a Stock detail should correspond to a product.

The main difference between the Bachman Notation and the Peter Chen notation is in the way [relationships between the entities](#) are represented.

There are many ER diagramming tools some of which comes with the commercial DBMS and some of which are open source. Some free software ER diagramming tools that can interpret and generate ER models, SQL and do database analysis are MySQL Workbench and DBDesigner.

Relationships between the Entities

The different entities in the database may have relationship with each other. It describes the association between the entities.

For example a Company can produce many products and the Products may belong to many categories. If we try to explain the relationship in words, it may take time and it may cause confusion to at-least some people. The author of the document may have to spent time explaining the different parts.

Hence we use standard notations to represent the relationships.

The relationships are classified according to their optionality and cardinality.

According to optionality they can be classified as







1. Optional relationship: This means that there may be instances in the first entity which may not have any instances in the second entity. The example is the Employee entity and the spouse Entity. There may be some employees who are not married who do not have any corresponding instance in the Spouse Entity.
2. Mandatory Relationship: This means that for every instance of the first entity there will be at-least one instance of the second entity. We can take the example of the Employee entity and the qualification entity of a Company. For every employee instance there should be at-least one instance of qualification since all employees should have a qualification.

The term cardinality in [Relational model](#) means the number of rows in a table (Relation). The cardinality of a relationship is the number of instances of the second entity that corresponds to each instance of the first entity.

According to Cardinality the relationships can be classified as:

1. One to One: The one relationship between the product and the Stock is a One to One relationship where each Product many have a Stock detail. But it is not necessary that all products in the Product Entity should have a Stock detail.
2. One to Many: This means that for each instance in the first entity there can be more than one instances in the second entity. If we take the example of the Company entity and the Product entity we can say that each company entity can have more that one corresponding Product instances in the Product entity. That is one company can have many products.
3. Many to Many: This means that each instance in the first entity can have more than one instance in the second entity and each entity in the second entity can have more than one instance in the first entity.

The relationships between the entities can be represented using the Bachman notation ot Peter Chen notation.

COMPONENT	REPRESENTATION	
NOTATION	PETER CHEN	BACHMAN
RELATIONSHIP		
CARDINALITY		
OPTIONALITY		

In Peter Chen notation, the optional relationship is represented using number zero and mandatory relationship is using numbers 1 and above or general notation of 'N'.

In Bachman notation, the mandatory relationship is represented using '1' on the relationship line and optional relationship using 'o' in the relationship line.

Different types of databases:

There are different types of databases based on the different [Data Models](#) based on which they are implemented. They are:

1. Hierarchical Database – This database is implemented based on the Hierarchical Model - example: IBM's IMS and Windows Registry
2. Network Database - This database is implemented based on the Network Model – example: IDMS (Integrated Data Management system)
3. Relational Database - This database is implemented based on the Relational Model- example: Oracle 8 and lower, DB2, MSSQL Server, MS Access
4. Object Relational Database – This database is relational database that are also object based. Many of the object oriented concepts like objects, reuse, overloading, encapsulation etc are implemented in these databases. ORDBMS object-oriented database concepts can be superimposed on relational databases. An object-oriented database interface standard is being developed by an industry group, the Object Data Management Group (ODMG) – example: Oracle 8i and higher, DB2 version 8 and higher, MSSQL Server 2000 and higher.

Relational Model

There were other models that were in use before the relational model.

Then why did we need another model?

They were the hierarchical model and the network model. But the relational model was the first model to be defined in formal mathematical terms. The operation of data in the relational database is by means of [relational algebra](#). The comparison of the relational model with the hierarchical and the network model lead some rigor being added to these models. But it could not be formalised beyond an extent due to the procedural nature of data manipulation in hierarchical and network model databases.

Another advantage of this model is that this model uses the values inside the relation itself to establish relationship with other relations. The network model and the hierarchical model uses pointers to do this. Due to these reasons, the relational model became the most popular model and it replaced the hierarchical and network model which were in use at that time. Today most of the databases are implemented based on it. Some existing systems still use the older models since the cost of migrating to relational database is prohibitively high.

Dr. E.F. Codd proposed a Data Model - the Relational model for database systems in 1970. It is the basis of RDBMS. In relational model the data is represented in the form of two-dimensional tables called relations. Let us understand it with the example of the store. Here we have to store the details of the products and the companies to which the products belong.

The relational model representation of these entities that we identified would be two tables.

Company Table:

Company Code	Name
112232	Colgate Palmolive
112233	PepsiCo
112234	Coco cola India
112235	Dhathri
112236	Meswak India
112237	Hindustan Lever

Product Table:

Product ID	Name	Price	Weight gms	Company ID
11290	Colgate Active Salt	45	200	112232
11291	Closeup Cool crystals	48	200	112237
11292	Pepsodent	40	200	112237
11293	Meswak	30	200	112236
11294	Mavila	35	200	112235
11295	Lays	10	100	112233

See the two tables and their data.

What are the features of relational model that you notice?

1. The data is represented using two dimensional tables
2. The [tables may related to each other](#)
3. The relationship is established using the values inside the tables.
4. The rows are called tuples and the columns are attributes
5. The field used to uniquely identify a tuple is called a [Primary Key](#)
6. The field in the second table used to establish the relationship with the first table is called the [Foreign Key](#)
7. Cardinality [of a table]: is the number of rows of a table
8. Degree [of a table]: is the number of domains from which the columns draw its value.

The example for the relational model is the company table with the attributes like company code and company name and the product table with the attributes like product id, product name, price, weight, company etc.

Properties of Relations

The body of a relation is basically a mathematical set.

Duplicate Tuples are not allowed: A relation cannot contain more than one tuple which have the same values for all the attributes. In any relation, every row is unique.

Tuples are unordered: The order of rows in a relation is immaterial. The tuples are maintained in the database based on internal logic of the DBMS.

Attributes are unordered: The order of columns in a relation is immaterial.

Values in the relation are atomic.

Some DBMS do maintain the order but when operating on the data, it becomes immaterial. Modern databases especially the ones used for data warehousing, store the attributes in the alphabetical order of the attribute name.

Values of the attributes are Atomic: Each tuple contains exactly one value for each attribute.

How is data interrelated in the database?

The [relationship is established between the different tables](#) in the database using the referential integrity constraints. The referential integrity constraints are Primary Key and the Foreign Key.

Primary Key is used to uniquely identify a row from the table. Remember that we all had ID numbers or roll numbers when we were in school. This number would uniquely identify a student in a class.

Foreign Key is a constraint that is applied on the child table in order to establish a relationship with the parent table. The foreign key always refers to the primary key of the parent table.

Let us take a look at some important features of the primary key and the foreign key.

1. Duplicate values cannot be added to a column marked as primary key
2. We can have only one primary key in a table.
3. The values in the primary key columns are generally not changed
4. A primary key can be referred by any number of foreign keys.
5. Foreign key values must match an existing value in the primary key column it is referring to or else be null.
6. A table can have more than one foreign keys referring to different primary key columns of different tables.

In the above example the columns that are involved in the referential integrity are CompanyCode, which is the primary key of the Company Table and Company ID of the Product table, which is the foreign key that refers to the CompanyCode of the Company table.

Relational Algebra

The relational algebra was made popular after Dr. E. F. Codd proposed the relational model in 1970. According to him the relational algebra was the basis of the query language.

Codd chose six primitive operators and a few derived ones for the relational algebra. The six primitive are:

[Selection](#)

[Projection](#)

[Rename](#)

[Set Operators](#)

Cartesian

Set Union

Set difference

The important ones among the derived operators are

Set Intersection

[Anti Join](#)

[Division](#) and

[Natural Join](#)

Selection (σ)

In relational algebra, Selection – also called restriction – is a unary operator which is written as $\sigma_{\text{aOb}}(R)$ OR

$\sigma_{a\Theta v}(R)$ where

Where,

- a and b are attribute names
- Θ is a binary operation in the set $\{\wedge, <, \leq, =, \geq, >\}$
- v is a value constant
- R is a relation

Here selection $\sigma_{a\Theta b}(R)$ selects all the tuples in R for which Θ holds between the attributes ' a ' and ' b '.

The selection $\sigma_{a\Theta v}(R)$ selects all the tuples in R for which Θ holds between the attribute ' a ' and value ' v '

For example, for the relation Product, the selection $\sigma_{Price>1000}(\text{Product})$ gives the set of tuples for which the price of the product is greater than 1000.

In SQL, selection operation is performed by using WHERE definition in SELECT, UPDATE, and DELETE statements. The selection condition can result in any of three truth values - true, false and unknown.

Projection (π)

In relational algebra, Projection is a unary operation written as $\pi_{a_1, \dots, a_n}(R)$. The result of the projection is a set which is obtained when all the tuples are restricted to a subset of all the available attributes.

In other words the projection is a vertical subset of the relation.

Rename (ρ)

Rename is a unary operation written as $\rho_{a/b}(R)$ where ' a ' and ' b ' are attribute names in the relation R .

The result of the rename is same as relation R with the difference that attribute ' a ' of the relation R is renamed to b .

For example $\rho_{Price/ProductPrice}(\text{Product})$ would rename the attribute Price as ProductPrice in the result.

Set Operators

The set operators given similar results as that of the mathematical set operators. But, though taken from the Set theory, there are some additional restrictions for the Union difference and Cartesian product.

- For For set union and set difference, the two relations involved in these Set Operations must be union-compatible—which means, the two relations must have the same set of attributes.
- For the Cartesian product to be defined, the two relations involved must have disjoint headers — which means, they must not have a common attribute name
- Since set intersection can be defined in terms of set difference, the two relations in this set involved in the set operation, intersection, must also be union-compatible.

Join and Join like operators

Natural Join

It is a binary operator written as $R \bowtie S$ where R and S are relations. The result of this natural join is the set of all possible tuples of R and S that are equal on attribute names.

The natural join is the relational counterpart of the logical AND, and it allows the combinations of relations that are associated by a foreign key.

For Example for the Relation Category {CategoryID, CatName} and Product{ProductID, ProductName, CategoryID} where CategoryID is the Primary key of the relation Category and the foreign key of the relation Product.

The result of Category \bowtie Product will have the effect of equijoin based on the attribute CategoryID.

Equi Join and Θ Join

Theta Join (Θ Join) is a binary operator written as $R \bowtie_{\theta} S$ or $R \bowtie_{\theta} S$ where ' a ' and ' b ' are attribute names, θ is a binary relation in the set $\{<, \leq, =, >, \geq\}$, v is a value constant, and R and S are relations.

For example Product{ProductID, ProductName, Price} and Range{Low, High, rangeName} are relations that do not have a shared attribute, where RangeName attribute can take values Premium, Medium and Economy. If we need a result of 'price range' for all the products, then we can use the theta Join $\text{Product} \bowtie_{>} \text{Range Price}$

BETWEEN LOW and HIGH. For a Θ Join the relations should be disjoint. That is there should not be shared attributes.

Equi Join is a binary operator and it combines two relations based on the equality of the values of a Shared attributes. That is when the operator Θ is an equality operator, it is called an equi join.

Anti Join

The antijoin is a binary operator and is written as $R \bowtie S$, where R and S are relations,. The result of an antijoin is only those tuples in R for which there is NOT a tuple in S that is equal on their common attribute names.

Division

It is a binary operator written as $R \div S$ and the result is all the combinations with tuples in S that are present in R.

Outer Join

It is a binary operator and it contains those tuples and additionally some tuples formed by extending an unmatched tuple in one of the operands by 'fill' values for each of the attributes of the other operand.

All the categories of outer joins assume the existence of a null value, ω , to be used for the 'fill' values.

There are three outer joins

Left Outer Join: The result of the left outer join is the set of all combinations of tuples in R and S that are equal on their common attribute names, in addition (loosely speaking) to tuples in R that have no matching tuples in S.

Right Outer Join: The result of the right outer join is the set of all combinations of tuples in R and S that are equal on their common attribute names, in addition to tuples in S that have no matching tuples in R.

Full Outer Join: combines the results of the left and right outer joins.

SQL

SQL or Structured Query Language is the popular querying language for managing data in relational database. It is one of the many querying languages that are in use. It is based on [relational algebra](#). SQL is one of the first languages for relational model.

SQL was developed at IBM by Donald D Chamberlin. The first version was used to query IBMs original relational database System R and was named SEQUEL. This was later changed to SQL since SEQUEL was the registered trademark of another company. The first commercial SQL was launched by Relational Software Inc (now Oracle Corporation) in 1979.

Based on the type of databases and the types of data sources that are being queried, there are different querying languages. Some of them are

.QL: is an object oriented querying language for relational database.

D: is a querying language for truly relational database.

DMX: is a querying language for data mining.

XQuery: is a querying language for XML data sources.

XSQL: is a querying language combining the power of XML and SQL.

YQL: is an SQL like querying language created by Yahoo!

Embedded SQL

Embedded SQL statements are SQL statements written inline within the source code of the host programming language. It combines the computing power of SQL and the data manipulating power of SQL.

The SQL standard defines embedding of SQL as 'embedded SQL' and the language in which SQL queries are embedded is referred to as the 'host language'. These statements are parsed by an embedded SQL pre-processor and converted to host language calls. Some of the popular pre-compilers are Pro*C, Pro*FORTRAN, Pro*COBOL etc.

The popular databases Oracle, SQLServer and DB2 support Embedded SQL. But MySQL does not support it and the support has been discontinued by SyBase.

In Oracle, the DBMS provides for the use of special variables called 'host variables' or 'bind Variables' that can be accessed by the host language and the DBMS. These variables are usually preceded by a colon. There is also a block declaration section which can be embedded within the host language to include a set of SQL statements.

DCL

DCL is the part of SQL which is used to control the access permission to the different objects in the database.

The statements in DCL are:

GRANT: used to grant permissions to the users

REVOKE: to revoke the permissions to the users.

How do we grant all the permissions to the users?

After creating a user account, we would need to grant all the permissions of this schema to the user. To do this for the user account 'Lily',

GRANT ALL ON SCHEMA to Lily;

To grant all the permissions of the Product table to user Lily, we can do this:

GRANT ALL ON Product TO Lily;

To grant only data retrieval permission,

GRANT SELECT ON Product TO Lily;

To grant retrieval permission to the user Lily, which this user can intern grant to other users,

GRANT SELECT ON Product TO Lily WITH GRANT OPTION;

To revoke the permission to delete from the Product table, for the user Lily

REVOKE DELETE ON Product FROM Lily;

DDL

To understand [what is DDL](#); let us take the example of a Library. How are books organized in it?

It has a section for books. In that section, there will be shelves to hold the related books together.

In the same way, in database also, there should be an object to hold the data. They are the 'Tables'. The table is the object, which defines the structure in which to hold the data in the database.

The part of the SQL Language used to define the structure of the Objects in the database is called the DDL (Data Definition Language).

The DDL is used to:

1. Define the structure using the [CREATE](#) statement.
2. Change the structure of existing objects using the [ALTER](#) Statement and
3. Remove an existing object using the [DROP](#) Statement from the database.

Create a Table

To create the tables in the database, we need to use the CREATE statement.

In the example of the store the tables identified during the design were the Company table and Product table.

Company Code	Name
112232	Colgate Palmolive
112233	PepsiCo
112234	Coco cola India
112235	Dhathri
112236	Meswak India
112237	Hindustan Lever

Product ID	Name	Product description	Price	Weight gms	Company Code
11290	Colgate Active Salt	ToothCare	45	200	112232
11291	Closeup Cool crystals	ToothCare	48	200	112237
11292	Pepsodent	ToothCare	40	200	112237
11293	Meswak	Aurvedic toothpaste	30	200	112236
11294	Mavila	Aurvedic toothpaste	35	200	112235
11295	Lays	Salt snack	10	100	112233

Let us look at the data. The Product ID is a whole number. Product name is set of characters (variable length character string). Product description is set of characters (variable length character string). Price is decimal number. Weight is decimal number. Company code is whole number. So the data that goes into the db should also match this type of data so as to avoid confusions and errors.

The data types that can be chosen for the attributes are:

Product ID – Number with five digits

ProductName - Varchar2 with a maximum length of 30 characters

ProductDescription – Varchar2 with a maximum length of 100 characters

Price – is a decimal number; with 5 digits before the decimal and two digits after the decimal. This is represented as Number (7, 2) and not Number (5, 2).

The basic syntax for the table is

```
CREATE TABLE table name
(
    Column1 datatype DEFAULT expr constraint,
    Column2 datatype DEFAULT expr constraint,
    ---
    ---
    ColumnN data_type DEFAULT expr constraint,
);
```

We can opt to create the table without constraints and add constraints at a later time; or create the table with constraints in the first instant itself. We will see both the

So the metadata identified for the table are:

1. The data types of the attributes
2. The restrictions on the values placed on the attributes. (The restrictions that are placed on the values that can go into a column are called the constraints)
 - a. The product id should – Primary Key (be unique value and should not be null)
 - b. The product name should be unique
 - c. The price cannot be left out (Cannot be null)

- d. Company code is the Foreign key referring to the Primary Key of the table Company

First creation of the table with constraints

```
CREATE TABLE Company
  (CompanyCode number Primary Key,
   CompanyName varchar2 (50) Unique,
  );
CREATE TABLE Product
(
  ProductID number Primary Key
  ProductName varchar2 (100) Unique,
  Price number not null,
  CompanyCode number references Company (CompanyCode)
);
```

CONSTRAINTS

Why do we need constraints?

When data is being added to the table, the applications business logic would need that certain restrictions be placed on the values that go into the table. These restrictions are necessary for the successful working of the application. The restrictions include, making some attributes mandatory, allowing only certain values to go into the attributes etc. Apart from these restrictions we would also need to maintain one to many relationship. We implement all these using constraints.

Constraints are used to define a data integrity rule. It is a rule that restricts the value for one or more columns in a table. Based on our need, different constraints can be set on the columns.

There are different constraints that can be used for a table. The different constraints used in are:

- [Not Null](#) – specifies that all the rows of the column should have a value. None of the rows in this column can have a null value. In other words values are mandatory for the column.
- [Primary Key](#) – The values a primary column should be unique and should not be null. A primary Key can be targeted by more than one foreign Keys.
- [Unique](#) – The values in a unique column should be unique
- [Foreign Key](#) – A foreign key constraint is used to establish relationship between two tables. It usually refers to the primary key column of the parent table.
- [References](#) – references is a constraint that works with the foreign key to refer to the primary key column of the parent table.
- [On Delete Cascade](#) – Is used to maintain the integrity of the data when a primary key value is removed from the parent table. If this constraint is used, the corresponding child rows are removed when a row is deleted from the parent table. This constraint works with the foreign key constraint.
- [On Delete Set Null](#) – This is similar to the On Delete Cascade. The difference is that instead of removing the rows containing the corresponding foreign keys, the corresponding foreign key values are set to null when a row is removed from the parent table.
- [Check](#) – this constraint is used restrict the values in the column of a table based on some values.

Let us understand the concept of constraint using the example of the Company table and the product table.

The constraint Primary Key is used to give a column the [characteristics of a Primary Key](#). If we want to make the ProductID the Primary Key of the Product table we it do like this along with the column syntax, while creating the table.

ProductID Number Primary Key,

The constraint, **UNIQUE** is used to impose a restriction on a column that the values in a column should be unique and no duplication is allowed. To give a 'Unique' constraint to the ProductName column, we can give this syntax while creating the table.

ProductName varchar2 (100) UNIQUE,

The **NOT NULL** constraint is used to make the insertion of values into a column mandatory – that is not allowing the insertion of [NULL](#) - while creating a row in a table. It is done by mentioning the NOT NULL constraint, after the data type of the column that has to be made mandatory, while creating the table.

Price Number NOT NULL,

How do we **establish a parent child relationship** between the company table and the product table? That is, a one to many relationship – One company can have many products.

This is done by using the referential integrity constraints, Primary Key and Foreign Key. The CompanyCode is primary key in the Company table and foreign key in the Product Table.

The foreign key can be given at the [column level](#) using the references constraint alone or at the [table level](#) using the constraints foreign key and references.

```
CREATE TABLE Product
(
    ProductID      Number (3)      Primary Key
    ProductName    Varchar2 (100) UNIQUE,
    Price          Number (7, 2)   NOT NULL,
    CompanyCode    Number (3)      REFERENCES Company (CompanyCode)
);
OR
```

```
CREATE TABLE Product
(
    ProductID      Number (3) PRIMARY KEY
    ProductName    Varchar2 (100) UNIQUE,
    Price          Number (7, 2)   NOT NULL,
    CompanyCode    Number (3),
    FOREIGN KEY (CompanyCode) REFERENCES Company (CompanyCode)
);
```

Though the column level and table level syntaxes are different, the effect is the same.

Now Let us see the need for Constraints ON DELETE CASCADE and ON DELETE SET NULL

Consider that both these tables have data and all parent rows have one or more child rows.

If we try to delete some values from the Company table which has child rows in the product table. You would see an error message stating that the row cannot be deleted.

Why do you think the database reacted like this?

1. There will be child records in the child table referring to the parent table's primary key.
2. If the deletion is allowed then there will be meaningless data in the child table

Hence the deletion is not allowed.

Let us assume the following restriction in the tables:

- a. The product id should – Primary Key (be unique value and should not be null)
- b. The product name should be unique
- c. The price cannot be left out (Cannot be null)
- d. Company code is the Foreign key referring to the Primary Key of the table Company
- e. The programmer wants the removal of a record and cascade its effect on the child table also; ie; remove the corresponding records from the child table also then this can be allowed if the constraint ON DELETE CASCADE was given in the child table as a continuation of the foreign key constraint.

The tables would be created in the following way.

```
CREATE TABLE Product
(
    ProductID      Number (3) Primary Key
    ProductName    Varchar2 (100) Unique,
    Price          Number(3)      NOT NULL,
    CompanyCode    Number (3) REFERENCES Company (CompanyCode) On Delete Cascade
);
```

If we had used the On Delete Set Null instead, It would have set the foreign key values to null when the parent key record is removed.

Suppose we want a business rule to be imposed on the value of an attribute. For example, we want to ensure the entire price data that goes into the table should be > 0. Or else the adding of the data should not be allowed. We can apply the CHECK constraint on the Price attribute.

```
CREATE TABLE Product
(
    ProductID      Number (3) Primary Key
    ProductName    Varchar2 (100) Unique,
    Price          Number (7, 2)   NOT NULL,
    CompanyCode    Number (3) References Company (CompanyCode) On Delete Cascade,
    Constraint Price_Check CHECK (Price>0)
);
```

Why did we give the check constraint at the end after all the columns?

Since the Price attribute already has another constraint. The check is the second constraint applied on the same attribute.

Suppose the Price attribute had only the check constraint it could be given like

```
CREATE TABLE Product
(
    ProductID Number (3) Primary Key
    ProductName Varchar2 (100) Unique,
    Price Number(7,2)      Constraint Price_Check CHECK (Price>0),
    CompanyCode Number(3) References Company (CompanyCode) On Delete Cascade,
);
```

What is Price_Check? It is the **name of the constraint** Check applied to the price attribute.

If the programmer does not give this name the database will generate a name. The name that the database generates will be in the format 'SYS0000##'.

It is always a good practice to name the constraints that you add to the table with meaningful and easy to remember names.

The use of constraint name is that the business rule which is added as a constraint can be removed at a later time if it is not required.

```
ALTER TABLE Product DROP CONSTRAINT Price\_Check;
```

Some features of constraints

- Constraints are added at the time of creation or by using the ADD clause of ALTER statement
- Constraints cannot be modified. It can only be added or dropped.
- Only violation to this rule is the Not Null constraint. It is added by using the MODIFY clause of the ALTER statement.

Till now we saw how to create a table and add constraints to it. But if we need to create a table based on the structure of an existing table, then what do we do?

That is; create a copy of the Products table and include the data also while copying.

We can use a CREATE statement with a subquery.

```
CREATE TABLE ProductsCopy AS SELECT * FROM Product;
```

When we create a copy of the table like this we need to remember that only the structure and the data is copied. The constraints of the base table will not be copied to the new table. If we need to add the constraints to the new table, then we need to add it explicitly using the ALTER statement.

Data types in Oracle

The data types as in any other programming language indicate the type of data that can be held. In a database we assign data types to the columns of the tables that we create. The data types ensure that only the right type of data goes into the column.

Let us now look at the different types of data types.

The different data types that are used in Oracle are basically classified as **Scalar data types** and **Composite data types**.

We also have special data types called the **LOBs** (Locator Objects) that can hold up to 4 GB. These types are stored outside the table once its size is bigger than 4000 bytes.

In addition to this we can create and store reusable types in oracle database called **the Abstract data types**. These custom data types can be used as data types of columns.

Let us see some of them in detail.

Scalar data types as the name suggest, hold a single value. The different types of scalar data types are:

[Character Types](#)

[Number Type](#)

[DateTime Types](#)

[Interval Types](#)

[Binary Types](#)

The following are the **character data types** that represent character strings:

CHAR (size): is used to store fixed length character strings. The size indicates the number of characters to be stored. This type can store a maximum of 2000 bytes.

For example, assigning Char (10) to a column would reserve 10 bytes for all the rows of this column. Even if we stored a value 'Lily' which is 4 bytes, it would be padded with 6 blank spaces and stored as 10 byte character string

VARCHAR 2(size): Varchar2 is used for storing variable length character strings. This type can store to a maximum of 4000 bytes.

If we assign varchar2 (10) to a column of a table, and store 'Lily' in one of the rows of this column, it would occupy only 4 bytes; unlike its char counterpart.

VARCHAR (size): In Oracle 9i, the meaning of Varchar is same as that of Varchar2. But the documentation insists that we use only Varchar2 for denoting variable length character strings since the meaning of Varchar may change in the future versions and this may cause problems during migrations to higher version.

NCHAR (size) and **NVarchar (size)**: All the characters in the English alphabet list can be represented using one byte. But this is not true to some languages like Chinese. So Oracle introduced the NChar and NVarchar types that can store characters as Unicode. The data stored as NChar and NVarchar can be retrieved and manipulated as any other type. NChar can hold up to 2000 bytes and NVarchar up to 4000 bytes.

LONG: type stores character strings up to 2GB. But this type has a restriction that you can have only one Long column per table.

The Numbers are represented by using the **Number type**

NUMBER (Precision, scale): type can store whole numbers and decimal numbers. The number type with one argument indicates a whole number with the number of digits as given in the precision. Precision takes values from 1 to 38. And scale from -84 to 127. A number type without argument takes the maximum size.

The **Date and time types** are:

DATE: is a for date fixed-sized 7 bit field that is used to store dates. Even though not indicated in the type, the time is also stored as part of the date. The default format for date is DD-MON-RR

TIMESTAMP (precision): is a variable-sized value ranging from 7 to 11 bytes and is used to represent a date/time value. The precision parameter determines how many numbers are in the fractional part of 'Second' field. The precision of the 'Second' field within the TIMESTAMP may have a value ranging from 0 to 9 and defaults to 6.

TIMESTAMP (precision) WITH TIME ZONE: is a fixed-sized value of 13 bytes and represents a date/time value along with a time zone setting. There are two ways to set the time zone. The first is by using the UTC offset, like '+10:0' and second by the region name, say 'Australia/Sydney'.

TIMESTAMP (precision) WITH LOCAL TIME: is a variable value ranging from 7 to 11 bytes. This data type is similar to **TIMESTAMP WITH TIME ZONE** with the difference that the data is normalised to the database time zone when stored and when the data is retrieved, users see the data in the session time zone.

The **Interval Types** are:

INTERVAL DAY (day_precision) TO SECOND (second_precision): is a fixed-sized 11 byte value that represents a period of time. It includes days, hours, minutes and seconds. The accepted values to both the precision fields are 0 to 9. The **second_precision** defaults to 2 and the **day_precision** defaults to 6.

INTERVAL YEAR (year_precision) TO MONTH: is a fixed-sized 5 byte value that represents a period of time. It includes years and months. The precision is the number of digits in the Year field. The accepted values are 0 to 9 and it defaults to 6.

Now let us look at the **Binary types**.

RAW (size): stores binary data upto 2000 bytes.

LONGRAW: stores binary data upto 2GB.

There is a special scalar data type

ROWID: is used to uniquely identify each row. It is this value which is returned by the [pseudocolumn](#) ROWID.

Composite data types can hold more than one value. The composite types are made by combining more than one scalar types or composite types. Examples of the composite types are record type, table type, Varray and nested table.

LOB

LOB types are special Locator Object types that can store up to 4GB. These types are stored inline – inside the table – up to a size of 4000 bytes, but are moved outside the table beyond this size and only the locators or pointers to its location are stored in the row and hence this name for the type.

There are four main locator objects:

CLOB: is used to store character strings up to a size of 4 GB. From oracle 9i onwards simple SQL can be used to manipulate and retrieve data from the CLOB.

NCLOB: stores character strings of NVarchar type up to 4 GB.

BLOB: stores binary data like images, sound etc up to 4GB. We need to use special functions to manipulate and retrieve data from the BLOB type.

BFILE: stores the locators to external files. Unlike the other LOB types, the external resources stored as BFiles cannot be manipulated through the database.

BLOB, CLOB and NCLOB are stored inline up to 4000 bytes after which it is removed to another part of the database and only the locators to the data are stored in the corresponding rows of this column. These types are stored inside the database. But BFiles are stored outside the database and hence cannot be manipulated through the database like the other LOBs.

ALTER

The alter statement is used to change the existing structure of a table.

We can use the ALTER statement to:

- Add a new column
- Modify an existing column
- Drop a column

Alter Statement to add, modify or drop columns

- ALTER TABLE table_name ADD Column_name data_type DEFAULT expr;
- ALTER TABLE table_name MODIFY Column_name data_type DEFAULT expr ;
- ALTER TABLE table_name DROP COLUMN Column_name;

In Oracle 9i we have a direct statement to rename a column using the alter statement.

```
ALTER TABLE Tablename RENAME COLUMN oldname TO newname;
```

When we create the table, we have the option of creating it with out the constraints and add it at a later time using the alter statement.

Step one: Create the table without constraints

```
CREATE TABLE Company
(
    CompanyCode Number (3),
    CompanyName Varchar2 (50),
);
```

Step Two: Add the constraints to the table using the ALTER statement.

Adding the constraint would mean the changing the definition of the table. Ie; we need to ALTER the table

```
ALTER TABLE Company ADD CONSTRAINT Comp_PK Primary Key (CompanyCode) ADD Unique
(Company Name);
```

Here the clause CONSTRAINT Comp_PK gives a name Comp_PK to the primary key of the Company table. Even though this clause is optional it is a good practice to use it.

To remove a constraint that we added we need to use the ALTER statement with the DROP clause

```
ALTER TABLE Company DROP CONSTRAINT Comp_PK;
```

We can add any number of ADD, DROP and MODIFY clause in the same ALTER statement. This is convenient since we do not have to repeat the ALTER clause for each task. We would just keep on adding the clauses to the right separated by space. But we need to keep in mind that all these changes should be to the same table. For example let us say we want to add a column Remarks to the Product table, remove the price column from the Product table and change the datatype of the product name to varchar (100).

We can write the syntax like this:

```
ALTER TABLE Product
    ADD Remarks Varchar2 (100)
    MODIFY ProductName Varchar2 (100)
    DROP COLUMN Price;
```

It is good to keep these **important points** in mind:

1. An ALTER statement with a MODIFY clause that mentions only the new data type would not remove any constraints or DEFAULT values set to it earlier. Constraints can be removed only by an explicit DROP clause with the ALTER statement.
2. When more than one clause is mentioned along with the ALTER statement, all the modifications should be to the same table.
3. The DEFAULT value set for a column cannot be removed. It can only be set to NULL by using the DEFAULT keyword again.
4. Constraints cannot be modified. It can only be added or dropped. But the NOT NULL constraint is added by using the MODIFY clause.
5. It is a good practice to use constraint name when adding the constraints.

DROP

If we need to remove the object that we created, then we need to use the DROP statement

```
DROP TABLE Product;
```

If we need to remove all the data of the table without removing the structure of the table then we need to use the TRUNCATE table statement.

```
TRUNCATE TABLE Product;
```

Though the TRUNCATE removes the data of the table, it is a DDL statement.

The basic difference between the TRUNCATE and the DML statement DELETE is that

- TRUNCATE is a DDL statement and DELETE a DML
- TRUNCATE has Auto-Commit working with it since it is a DDL
- TRUNCATE does not fire triggers but DELETE does.
- TRUNCATE releases the space reserved for the table, but DELETE does not.

RENAME

If we need to rename an object of the database for any reason, we can use the RENAME statement. It works for not just a table but for all the object of the database.

If we wanted to rename the ProductsCopy table as Products_BU then we can use the RENAME statement like this.

```
`      RENAME ProductsCopy TO Products_BU
```

So the general syntax of the RENAME statement is

```
RENAME oldObjectName TO newObjectName;
```

DML

The DML - Data Manipulation language- is the part of the SQL that is used to manipulate the data in the database.

After we have created the objects to hold the data we can add data and manipulate it. DML part of SQL helps us to do this.

Using DML we can

1. Add the data to the table using the [INSERT](#) statement
2. Change the existing data in the table using the [UPDATE](#) statement
3. Remove the existing data in the table using the [DELETE](#) statement
4. Retrieve data from the database using the [SELECT](#) statement

Insert Data into the table

To insert the Data in the database the **INSERT** statement of the DML is used

Suppose we want to add a row of data to the Company Table

112232	Colgate Palmolive
--------	-------------------

This is done by the INSERT Statement

```
INSERT INTO Company (CompanyCode, CompanyName) VALUES (112232, 'Colgate Pamolive');
```

The INTO clause indicates the table to which the data goes. This is followed by the list of columns separated by commas to which the data is added. The VALUES clause gives the list of values to be added to the columns in the same order as that specified in the INTO clause.

11290	Colgate Active Salt	ToothCare	45	200	112232
11291	Closeup Cool crystals	ToothCare	48	200	112237

```
INSERT INTO Product (ProductID, ProductName, ProductDescription; Price, CompanyCode) VALUES  
(11290, 'Colgate Active Salt', 'ToothCare', 45, 200, 112232 );
```

What will happen if the following query is executed?

```
INSERT INTO Product (ProductID, ProductName, ProductDescription; Price, CompanyCode) VALUES  
(11291, 'Closeup Cool crystals', 'ToothCare', 48, 200, 112237);
```

It will give error since there are no values added in the Company table with primary key value 112237.

Suppose this entry is made in the Company table

```
INSERT INTO Company (CompanyCode, CompanyName) VALUES (112237, 'HLL');
```

What will happen if the following query is executed?

```
INSERT INTO Product (ProductID, ProductDescription, CompanyCode , ProductName, Price,) VALUES  
(11291, 'Closeup Cool crystals', 'ToothCare', 48, 200, 112237);
```

Error: Since the order in which the values are mentioned does not match with the datatype of the attributes.

What will happen if the following query is executed?

```
INSERT INTO Product VALUES (11291, 'Closeup Cool crystals', 'ToothCare', 48, 200, 112237);
```

This will work since the values are given in the same order as the order of attributes in the table

Notice that all the character string data has to be enclosed within single quotes. Otherwise it will be interpreted as column names and would lead to error. In all SQL statements we need to remember that character string and dates are always enclosed within single quotes.

UPDATE

Suppose I want to change the name of the company from HLL to Hindustan Unilever Limited. For this we would use the UPDATE statement

```
UPDATE Company SET CompanyName = 'Hindustan Unilever Limited' WHERE CompanyCode =  
112237;
```

How will you change the price of Close up Cool Crystals to 52?

UPDATE Product SET Price = 52 WHERE ProductID = 11291

DELETE

Suppose the store stopped dealing with the product Close Up Cool Crystals and it needs to be removed from their database. What will you do?

For removing existing data from the database, use the Delete statement.

DELETE FROM Product WHERE ProductID = 11291;

What does the following statement do?

DELETE Products;

It will delete all the product details.

If we have a table Product_Copy that has the same structure as the Product table but does not have its data. Then we can copy the data of Product to Product_Copy by using the INSERT statement with a subquery.

The statement is used like this:

INSERT INTO Product_Copy SELECT * FROM Product;

SELECT

The data, which is stored in the database, has to be retrieved for the different purposes. If we take the example of a store,

1. The storeowner needs know the list of products.
2. List of Companies he is dealing with
3. The list of products below the price of 50

For retrieving the data from the database in the needed format, the SELECT statement is used

Clauses in the SELECT statement:

The clauses are the different important parts of the SELECT statement that has a specific function.

The different clauses are:

[SELECT](#)

[FROM](#)

[JOIN](#)

[WHERE](#)

[GROUP BY](#)

[HAVING](#)

[ORDER BY](#)

The **SELECT clause** specifies what to select. The SELECT keyword is followed by the list of columns whose values are to be retrieved. The select clause is followed by the **FROM clause** which specifies from where to retrieve the values.

The simplest SELECT statement looks like this:

SELECT * FROM Product;

It retrieves all the data of all the columns in the Product table.

These are more specific SELECT queries. What do they do?

These queries retrieve all the rows of some specific columns in the Product table, or the vertical subset of the table.

SELECT Productname FROM Products;

SELECT CompanyName FROM Company;

This query retrieves some rows of some of the columns of the Product table. That is the combination of vertical and horizontal subset.

SELECT Productname FROM Products WHERE Price<50;

So to obtain the **vertical subset** we explicitly mention the column names in the SELECT clause and to obtain the **horizontal subset** we filter the rows using the WHERE clause.

General Syntax of the SELECT query is:

SELECT * | {DISTINCT column [, column]} FROM Tablename;

Suppose that we need to know the type of products available in the store

We chose to write the select query like this:

```
SELECT ProductDescription FROM Products;
```

This query would give the result like this.

Product description
ToothCare
ToothCare
ToothCare
Aurvedic toothpaste
Aurvedic toothpaste
Salt snack

But this is not the result that we expected. We need to see the list of product descriptions without repetition. So we need to rewrite the query as:

```
SELECT DISTINCT ProductDescription FROM Products;
```

Now we get the result as.

Product description
ToothCare
Aurvedic toothpaste
Salt snack

DISTINCT Clause is used to retrieve the data of a column without repetition.

The SELECT Query is by default ALL. “**ALL**” clause would give all the values including the repeating values.

We use the **WHERE Clause** when we want to retrieve the data that satisfied a certain criteria or condition,

“**WHERE** clause” is used to narrow down the query

While retrieving the data we can do a lot of operation on the data.

Operators in SQL

There are operators in SQL also as in any other programming language. These operators fall into six groups.

The different operators that can be used in SQL are:

- [Arithmetic](#)
- [Comparison](#)
- [Logical](#)
- [Concatenation](#)
- [Set](#)
- and [Special operators](#).

Arithmetic Operators

The arithmetic operators are used to do arithmetic operations on the data while using the SQL statement. The

Arithmetic operators that can be used

plus (+)
minus (-)
divide (/)
multiply (*)

Let us understand arithmetic operators through an example:

If the store owner wanted to apply a discounted price on the products of the Company with CompanyCode 112237, He can find the discounted price as

```
SELECT Productname, Price*0.9 “Discount Price” FROM Product where CompanyCode=112237;
```

The result will be:

Name	Product description	Discount Price
Colgate Active Salt	ToothCare	40.5
Closeup Cool crystals	ToothCare	44.2
Pepsodent	ToothCare	36
Meswak	Aurvedic toothpaste	27
Mavila	Aurvedic toothpaste	31.5
Lays	Salt snack	9

What effect did the phrase “Discount Price” have in the result?

```
SELECT Productname, Price*0.9 “Discount Price” FROM Product WHERE CompanyCode=112237;
```

It changes the heading of that particular result. It is called the Column Alias. The Column alias is use to give a heading to the column result which is different from the column heading in the table. If we do not give a column alias, the name of the column in the table itself will be the heading of the column in the result.

If we had written the column alias using an underscore like this query instead of a space then the double quotes are not needed,

```
SELECT Productname, Price*0.9 Discount_Price FROM Product where CompanyCode=112237;
```

Comparison operators that can be used

Equal (=)

Greater Than (>) and

Greater Than or Equal To (>=)

Less Than (<) and

Less Than or Equal To (<=)

Inequalities (< > or !=)

What are the results of these queries?

```
SELECT ProductName WHERE ProductName > ‘
Ariel’
```

There are some special operators used in SQL:

- LIKE Match a character pattern
- BETWEEN Between 2 values(inclusive)
- IN(set) Match any in a set of values
- IS NULL is a null value

The **LIKE** Operator is used to match a pattern.

The **wild cards** ‘%’and ‘_’work with the LIKE operator. ‘%’ is used to indicate a set of positions that can be of any character and of any number. ‘_’ is used to indicate a single position of any character.

If the Store owner wanted

1. the list of products beginning with H
2. List of products that has i as the second letter

```
SELECT ProductName
FROM Product
WHERE ProductName LIKE ‘H%’;
SELECT ProductName
FROM Product
WHERE ProductName LIKE ‘_i%’;
```

If we want Products between the price range 500 and 1000 we use the **BETWEEN**.

```
SELECT ProductName
FROM Product
WHERE Price BETWEEN 500 and 1000;
```

The “BETWEEN” keyword includes the range that is specified also. That is the above SELECT statement has the same effect as:

```
SELECT ProductName
FROM Product
WHERE Price >=500 and Price <=1000;
```

If we want the products with price 200 or 100 or 400 or 500 or 1000; we use the **IN** operator.

```
SELECT ProductName
FROM Product
WHERE Price IN (200, 100, 400, 500, 1000);
```

It has the same effect as :


```
SELECT ProductName
FROM Product
WHERE Price =200 OR Price =100 OR Price =400 OR Price =500 OR Price =1000;
```

To retrieve the list of products for which weight is not mentioned

```
SELECT ProductName from Product WHERE Weight IS NULL;
```

Logical Operators

- AND Returns TRUE if both components are true
- OR Returns TRUE if either of the condition is true
- NOT Returns TRUE if the following condition is false

If we want the products having price >100 and from company =112237;

```
SELECT ProductName FROM Product WHERE Price>100 AND CompanyCode = 112237;
```

Here we used the logical AND operator to combine the two conditions and data will be retrieved only if both the conditions will be satisfied.

Concatenation operator (||)

The concatenation operator is used to concatenate the character strings or data. It has the same effect of the concatenate function

If we wanted to retrieve the data as

```
"      Product name and price
      Close Up Cool Crystal Costs Rs. 48
      Colgate Active Salt costs Rs. 50
      ..."
```

```
SELECT Productname|| 'Costs Rs.' || Price "Product name and price" FROM Product;
```

We use the concatenation operator '||'.

Colleague

SET Operators

You would be familiar with the set operations that you learned in your school for mathematics. These SET operations can be done in SQL also using SET operators. The meanings of these SET operators is similar to the meanings of the SET functions that you learned.

They are:

[UNION](#)

[UNION ALL](#)

[INTERSECT](#)

[MINUS](#)

The **UNION** is used to combine the outputs of two SELECT queries. For example,

```
SELECT ProductName FROM Product;
```

```
UNION
```

```
SELECT ProductName FROM ProductCopy;
```

The output of this query would be the UNION of the ProductNames from the Product table and the ProductCopy table. Here the duplicate values in the two Sets are not repeated.

The **UNION ALL** is similar to UNION, but the difference is that if we use UNION ALL the duplicate values are also displayed.

INTERSECT gives the common elements in the output of the two queries. It is used like this:

```
SELECT ProductName FROM Product;
```

```
INTERSECT
```

```
SELECT ProductName FROM ProductCopy;
```

Here the output would be the common elements in the results returned by the two SELECT queries.

MINUS also has the same meaning as in the mathematical set operation. Let us see a query.

```
SELECT ProductName FROM Product;
```

```
MINUS
```

```
SELECT ProductName FROM ProductCopy;
```

The output of this query would be the Products that are in Product table that are not in the ProductCopy table.

Rule of precedence for the different operators in SQL

- Arithmetic
- Concatenation
- Comparison
- IS [NOT] NULL, LIKE, [NOT] IN
- [NOT] BETWEEN
- NOT
- AND
- OR

ORDER BY

“The **ORDER BY** Clause is used to sort the retrieved data based on the value of an attribute”

The sort can be done for numeric and character data in the ascending or in the descending order.

The Order By clause should be followed by the column based on which the data should be sorted and then by the keyword ASC for ascending order or DESC for the descending order. The default order of sort is ascending order. So if the sort order is not specified after the ORDER BY clause, the data will be sorted in the ascending order of column mentioned in the ORDER BY clause.

Let us take an example.

Suppose the storeowner wanted the product list in a sorted order

1. In the alphabetical order
2. In the descending order of price

The query can be written like this:

```
SELECT Productname, Price FROM Product ORDER BY ProductName;  
SELECT Productname, Price FROM Product ORDER BY Price DESC
```

What is the effect of having two column names in the ORDER BY clause?

```
SELECT Productname, Price FROM Product ORDER BY ProductName, Price DESC;
```

A query using all these

```
SELECT Productname, Price FROM Product WHERE Companycode =112237 ORDER BY ProductName;
```

A query like this,

```
SELECT Productname, Price FROM Product WHERE Companycode =112237 ORDER BY 1;
```

May not seem to be right, BUT this executes perfectly and gives the ProductName and Price sorted in the ascending order of the first column in the SELECT clause. In this case the ProductName.

BUT, this statement would give you an error since there is no ‘third column’ in the SELECT clause.

```
SELECT Productname, Price FROM Product WHERE Companycode =112237 ORDER BY 3;
```

Let us analyze one more case.

The columns that we use in the ORDER BY clause can be columns in the SELECT clause or columns in the table that are not part of the SELECT clause.

That is;

```
SELECT Productname FROM Product WHERE Companycode =112237 ORDER BY Price;
```

Is perfectly fine and would give you the names of the products sorted in the ascending order of price.

NULL

How is NULL handled in the database?

NULL in database denotes that there is no relevant data or the absence of information.

It has the following features:

- It is not same as numeric zero or character string "NULL" or "".
- It will be treated as the biggest value and will appear at the end of the list in ascending order and beginning of the list in the descending order.
- It can be assigned but cannot be equated with any value, not even itself.
- A NULL is never equal to another NULL and hence the column to which [Unique constraint](#) is applied can take any number of NULL values.
- Any arithmetic operations with NULL will result in a NULL value.

Example:

The following statements that use NULL in the WHERE clause like this are meaningless.

```
SELECT COUNT (*) FROM Product WHERE NULL = NULL;
```

```
SELECT COUNT (*) FROM Product WHERE NULL <> NULL;
```

```
SELECT COUNT (*) FROM Product WHERE NULL = '';
```

It can be used in INSERT statements to denote that there is no information in a row of a column.

```
INSERT INTO Product (ProductID, ProductName, ProductDescription)
VALUES(10, CP, NULL);
```

It can be assigned to any rows of a column

```
UPDATE test SET test1 = NULL WHERE ROWNUM = 1;
```

The NULL cannot be equated to or compared to using the equality operator (=). A special operator [IS NULL](#) is used to compare a value to NULL. Let us see a sample query that does a comparison with NULL.

```
SELECT COUNT (*) FROM Product WHERE ProductName IS NULL;
```

There are some [special functions that involve NULL](#) values available in Oracle.

JOIN

Why is there a need for Joins?

In the database the data needed for an application is distributed among different tables. This is so based on the normalisation principles. So when we need to retrieve the data back in the form of a comprehensive report, we may need to take the data from two or more tables so as to get a full picture. It is the concept of Joins that enable us to implement this multi table retrieval.

The **JOIN** is a Keyword that helps to retrieve data from more than one table. Depending on the type of type of output that we need and the Join condition, there are different types of Joins.

Depending upon the report that we need to retrieve from the database, the Joins are of **different types**:

1. [Equi Join](#) or Inner Join
2. [Outer Join](#)
3. [Non- Equi Join](#)
4. [Cross Join](#)
5. [Natural Join](#)
6. [Mutable Join](#)

To understand the concept of Joins let us take the example of the details of the category and the products in the category. Since the concept of Joins is slightly confusing to some candidates, I have chosen to go clause by clause in the explanation of the [Equi Join](#). So if you have difficulty with Joins make sure that you read this first.

Category Table

CategoryID	CategoryName	Description
C10	Tooth Care	Tooth care product
C20	Skin Care	Skin care product
C30	Stationary	Stationary Item
C40	Food	Food Product

Product Table

ProductID	ProductName	Price	Category ID
P10	CP	45	C10
P20	PDT	48	C10
P30	VVL	10	C20
P40	PML	10	C20
P50	PRS	15	C20

Range Table

RangeName	LowerRange	HigherRange
Low	0	499
Meduim	500	999
High	1000	5000

Information Given:

- The CategoryID is the [primary key](#) of the Category table.
- The ProductID is the primary key if the Product table
- The CategoryID in the Product table is the [foreign key](#) referring to the categoryID (Primary Key) of the Category Table.
- [Database](#) used is Oracle 9i

EQUI JOIN (INNER JOIN)

The Join that is used to retrieve the details from a table and the corresponding data from another table, only when there is a match in the [join condition](#) and the join condition involves an equality check is called an Equi Join.

The example used to demonstrate the Equi Join

Suppose that we need a report as given below:

ProductName	Price	CategoryName	Description
CP	45	Tooth Care	Tooth care product
PDT	48	Tooth Care	Tooth care product
VVL	10	Skin Care	Skin care product
PML	10	Skin Care	Skin care product
PRS	15	Skin Care	Skin care product

Here we need the product details and the corresponding category details. Let us take a look at the Product table once again. To which category does the product CP belong to?

You are right if you gave the answer as "Tooth Care". How did you find it? You took the category ID of the product CP that is C10. Then you went to the Category table and searched for the ID C10. When you found the match, you took the Category name corresponding to the ID C10. And that was "Tooth Care"

We would use the same logic in the Join condition.

Let us now translate our logic to SQL. First write the SELECT clause for specifying the columns that you want to retrieve

```
SELECT ProductName, Price, CategoryName, Description
```

Now add the FROM Clause. But here we need to mention two tables.

```
SELECT ProductName, Price, CategoryName, Description
```

```
FROM Category JOIN Product
```

Now we need to specify the Join condition that we mentioned earlier since in the ANSI syntax, the **JOIN** keyword should be followed by the Join condition. The Join condition is the logical condition based on which the data in the one table is related to the second table. This would be given in the ON clause

In the product and Category table, the data is related to each other based on the primary key – foreign key relationship; that is referential integrity.

```

SELECT ProductName, Price, CategoryName, Description
FROM Category JOIN Product
ON (Category.CategoryID = Product.CategoryID);

```

This means that retrieve the details specified in the SELECT clause wherever there is a match in the categoryID of the Product table and the category Table. We used the table name to differentiate between the CategoryID of the product table and the category table since; the two columns are having the same name.

To introduce one more concept, let us make an assumption that the column names of the ProductName and CategoryName columns were same; say, 'Name'. Then how would you write the query.
Will this work?

```

SELECT Name, Price, Name, Description
FROM Category JOIN Product
ON (Category.CategoryID = Product.CategoryID);

```

No, you would get a column ambiguity error for the Name column as is the case if we had not used the table name to differentiate the CategoryIDs in the ON clause.

We can write the query as:

```

SELECT Product.Name, Price, Category.Name, Description
FROM Category JOIN Product
ON (Category.CategoryID = Product.CategoryID);

```

Alternatively we can use the **table alias** like this.

```

SELECT p.Name, Price, c.Name, Description
FROM Category c JOIN Product p
ON (c.CategoryID = p.CategoryID);

```

Table alias is a short alias given to a table used in the FROM clause of a SELECT statement. is used to reduce the length of the query and also to make it more readable. The scope of the table alias is only within the query.

Since the names of the columns involved in the Join condition are the same, the above Join statement can also be written using the USING keyword as:

```

SELECT ProductName, Price, CategoryName, Description
FROM Category JOIN Product
USING (CategoryID);

```

Here the database itself will generate the Equi Join condition based on the columns given as the argument for the USING clause.

Keep in mind that if you need to use the column used in the USING clause in the SELECT clause, it **should be used without** the [table alias](#) even though the names of the two columns in the two tables are the same.

Database would solve the ambiguity issue for this column.

So we would write the query like this:

```

SELECT ProductName, Price, CategoryName, CategoryID, Description
FROM Category c JOIN Product p
USING (CategoryID );

```

The **Oracle 8i** does not use the Keywords like JOIN, ON and USING. Here the table names are separated by commas and the Join condition is mentioned in the WHERE Clause:

```

SELECT ProductName, Price, CategoryName, Description
FROM Category c, Product p
WHERE (c.CategoryID = p.CategoryID );

```

OUTER JOIN

The join that is used when we need all the rows from one table and the corresponding rows from a second table regardless of whether there is a match or not. For rows in the first table that has corresponding data in the second table the data is shown and for rows that do not have a report, [NULL](#) is displayed.

There are different types of outer Joins.

1. Left Outer Join – Gives all the rows of the
2. Right Outer Join
3. Full Outer Join

To understand outer Joins let us take this example. Suppose we need this report.

CategoryName	ProductName	Price
Tooth Care	CP	45
Tooth Care	PDT	48
Skin Care	VVL	10
Skin Care	PML	10
Skin Care	PRS	15
Stationary		
Food		

Here, for the categories that has products, the product details are displayed, but for products that do not have corresponding products in the Product table nothing is displayed.

The SELECT query is written using the Outer Join like this:

```
SELECT CategoryName, ProductName, Price
      FROM Category LEFT OUTER JOIN Product
            USING (CategoryID);
```

Here we need all the rows from the Category table, which is in the left side of the Join. So we used the Left Outer Join.

We mentioned the Category table first, which is the smaller table of the two due to [performance](#) reasons. In Oracle 9i we use the keywords like LEFT OUTER JOIN, RIGHT OUTER JOIN and FULL OUTER JOIN since it follows the **ANSI Syntax**. Here if the programmer forgets to give the Join condition, it gives a syntax error. But Oracle 8i does not follow ANSI syntax and here if we forget the Join Condition in the WHERE clause, it gives a Cartesian product.

These keywords are not there in **Oracle 8i** since Oracle 8i does not follow the ANSI syntax.

Syntax for Left Outer Join in Oracle 8i is:

```
SELECT CategoryName, ProductName, Price
      FROM Category c, Product p
            WHERE (c.CategoryID=p. CategoryID (+));
```

The (+) on the right side of the Join condition tells the DBMS that we need a Left Outer Join. If the (+) was in the left side we would get the Right Outer Join.

```
SELECT CategoryName, ProductName, Price
      FROM Category c, Product p
            WHERE (c.CategoryID (+) =p. CategoryID);
```

And the Full Outer Join is given by the union of the Left and the Right Outer Joins.

```
SELECT CategoryName, ProductName, Price
      FROM Category c, Product p
            WHERE (c.CategoryID=p. CategoryID (+));

UNION

SELECT CategoryName, ProductName, Price
      FROM Category c, Product p
            WHERE (c.CategoryID (+) =p. CategoryID);
```

NON-EQUI JOIN

The Join that is used to retrieve data from two tables based on a Join condition that uses a condition other than an equality check is called a Non-Equi Join.

Consider that the report that the store owner needs is this.

RangeName	ProductName	Price
Low	CP	45
Low	PDT	48

Low	VVL	10
Low	PML	10
Low	PRS	15

We need the price range of the product along the product list. So the tables that we need to take from are [Product table](#) and the [Range Table](#). The product range is found by taking the price of the product, whether it comes in between any of the LowerRange and the HigherRange in the Range table. Whenever the condition is met, retrieve the RangeName.

Here condition check that is used to retrieve the corresponding data involves the between operator.

So we would use the Non-Equi Join. We do not have a separate Keyword for this Join. The only thing is that the join condition does not involve an equality check

```
SELECT RangeName, ProductName, Price
      FROM Range JOIN Product
      ON (Price BETWEEN LowerRange and HigherRange);
```

Oracle 8i Syntax:

```
SELECT RangeName, ProductName, Price
      FROM Range, Product
      WHERE Price BETWEEN LowerRange and HigherRange;
```

CROSS JOIN

A cross Join is a join where all the rows of one table is combined with all the rows of the second table. The Cross Join does not have a Join Condition.

In Oracle 9i the JOIN keyword should be followed by a Join condition. But for Cartesian product, we do not have a Join condition. Hence we have a special type of Join in Oracle 9i known as the Cross Join.

There may be situations where we need a Cartesian product - all the possible combinations of all the rows of one table with another. That is there is no need of a join condition here since all the rows are to be combined with all the rows of the other table.

```
SELECT CategoryName, ProductName FROM Category CROSSJOIN Product;
```

Oracle 8i does not have a special Join called Cross Join. It gives the Cartesian product if we simply omit the Join condition.

```
SELECT CategoryName, ProductName FROM Category, Product;
```

NATURAL JOIN

A Natural Join is used when we need an Equi join between two tables and only the columns involved in the Join condition are having the same name. The DBMS would take the two columns having the same name and automatically generate an equijoin condition based on that column. Since the Equi Join condition is automatically generated, we do not have to mention the Join Condition along with the Natural Join Keyword.

In the Category and Product tables of the database for the store, only the columns involved in the Join condition (CategoryID of the [Product Table](#) and the CategoryID of the [Category Table](#)) have the same name.

Hence the natural Join can be used if we need an EquiJoin between these two tables.

```
SELECT CategoryName, ProductName FROM Category NATURAL JOIN Product;
```

Notice that we do not have to mention the Join Condition since the Join condition is automatically generated by the DBMS based on the columns having the same name in the two tables in the FROM clause.

There is no Natural Join in Oracle 8i.

MUTABLE JOIN

We use the mutable Join when we need to take data from more than two tables.

Suppose that we need a report like this:

ProductName	Price	RangeName	CategoryName
CP	45	Low	Tooth Care
PDT	48	Low	Tooth Care
VVL	10	Low	Skin Care
PML	10	Low	Skin Care

PRS	15	Low	Skin Care

Here Product Name and Price is from the [Product table](#), the Range from the [Range table](#) and the CategoryName from the [Category Table](#).

Here we need to Join two of the tables first and to that join the third table.

```
SELECT ProductName, Price, RangeName, CategoryName
      FROM Category
      JOIN Product
      USING (CategoryID )
      JOIN Range
      ON (Price BETWEEN LowerRange and HigherRange);
```

Oracle 8i Syntax

```
SELECT ProductName, Price, RangeName, CategoryName
      FROM Category c, Product p, Range
      WHERE (c.CategoryID=p.categoryID)
      AND (Price BETWEEN LowerRange and HigherRange);
```

SELF JOIN

We are now familiar with how to establish relationship between tables in a relational model. But sometimes we may need to have certain simple relationships within a table itself. For example an employee manager relationship can be represented within a table itself if we do not have any more data connected to the relationship.

To retrieve data based on a relationship within the table, we use the SELF JOIN.

A Self Join is a Join in which a table is Joined to itself so that we can retrieve the data corresponding to the relationship. It can also be used to check the internal consistency of data.

EmplyID	ManagerID	Name	Salary
100		Mark	60000
101	100	Mary	40000
102	100	Mathew	35000
103	104	Muhammed	55000
104		Majnu	46000
105	104	Madhu	46000

Here we have a relationship represented within the table itself - Who is whose manager. This is represented using an additional column named ManagerID.

Consider that we need the names of employees and managers. There is no straight forward way to retrieve it from the table.

Now look at the sample data and find out who is Madhu's manager.

It is Majnu. What logic did you use to find this out?

The logic that we used is – we took the managerID of Madhu and compared it with the employeeID of other employees. It matched the employee id of Manju.

So we can generalize the relationship as Employee's ManagerID = Manager's EmployeeID. This is the condition that we are going to use to retrieve the names of employees and managers.

Here since the relationship exists within the table itself, we would use the SELF JOIN. That is; the table is joined to itself.

Here we need to keep in mind that two different table alias need to be used while Joining the table to itself.

So we write the query like this:

```
SELECT Emp.Name "Employee Name" , Mgr.Name "Manager Name"
      FROM Employee Emp JOIN Employee Mgr
      ON (Emp.ManagerID = Mgr.EmplyID);
```

Here we used the table alias to differentiate between the Employee name and the manager name since it is basically one single column.

How does this work?

Here two copies of the same table would be stored in the database memory with two different alias names. A row is retrieved from one – in this case from the alias; Emp– and the ManagerID of that row is compared to the EmployeeID of all the rows in the Mgr alias. When there is a match, the name is retrieved as the manager name- that is; Mgr.Name.

To fully understand the concept let us also look at the output of the query.

Employee Name	Manager Name
Mary	Mark
Mathew	Mark
Muhammed	Majnu
Madhu	Majnu

But here we notice that the names of the managers are not included in the output. Why is that?

This is because, the join condition “Emp.ManagerID = Mgr.EmployeeID” is not satisfied for the managers since their managerID is blank. So if we need the names of employees who are also managers in the output, then [equi join](#) condition in the Self Join will not suffice.

How do we include the name of the manager also in the output?

That is; include all the employee names even though it does not satisfy the join condition.

Here we can use the [Outer Join](#) for this result.

```
SELECT Emp.Name “Employee Name”, Mgr.Name “Manager Name”  
FROM Employee Emp LEFT OUTER JOIN Employee Mgr  
ON (Emp.ManagerID = Mgr.EmployeeID);
```

Functions

Functions are inbuilt logic stored as objects in the database which can be invoked whenever we need that logic. It may take one or more arguments and it always returns a value.

Suppose we have a piece of code to find the sum of two numbers. If we need to reuse it a lot of times, how will we make this piece of code reusable?

Like in other programming languages, we can implement it as a function and invoke it wherever this logic is needed. The database has a lot of predefined functions and we can use it according to our need.

So the different types of predefined functions are:

[Aggregate functions](#) (Group functions)

[Date and time functions](#)

[Arithmetic functions](#)

[Character functions](#)

[Conversion functions](#)

Miscellaneous functions

Aggregate functions (Group functions)

These functions take the values of a group of rows and give a single result. Hence it is called the **Multiple Row Functions** or **Group Functions** or **Aggregate Functions** also. They take a column name as argument. Aggregate functions are called group functions since they work with the GROUP BY clause to operate on a group of data.

The common aggregate functions are:

[Count](#) – returns the number of rows retrieved by the select query.

[Average](#) – returns the average value for all the values in the column

[Sum](#) - returns the sum of all the values in a column

[Min](#) - returns the minimum value in a column

[Max](#) - returns the maximum value in a column

StdDev - returns the standard deviation of the values in a column

Variance - returns the variance of the values in a column

Consider the functionalities like Count, Average, Sum, Minimum, Maximum etc. What is the common feature of these functions?

1. It processes a set of values
2. Gives a single result

How can we find the Average price of toothcare products?

SELECT **AVG** (Price) FROM Products WHERE ProductDescription = 'Tooth Care';

What is the output of the following Queries?

SELECT **Count** (*) From Products;

This query returns the number of rows in the Product table. We can also say that this query returns the number of products.

SELECT **Max** (Price) FROM Product;

This query returns the Maximum price in the Product table.

SELECT **Min** (Price) FROM Product WHERE ProductDescription = 'Tooth Care';

This query returns the Minimum price in the Product table for the products with description "Tooth care.

SELECT **Sum** (Price) FROM Product;

This query returns the Sum of all the prices in the Product table.

Date functions

The date functions are used to manipulate the dates.

Add_months: adds a number of months to a specified date

SELECT Task, Startdate, Enddate ORIGINAL_END, **ADD_MONTHS**(Enddate,2) "Extended date" FROM Project;

Last_day: returns the last day of a specified month

SELECT Enddate, **LAST_DAY** (Enddate) FROM Project;

Months_Between: It takes two dates as the arguments and returns the months between the two dates as result. This is the only date function that returns a number. The date with higher value needs to be specified

first otherwise it would return a negative value.

```
SELECT Task, Startdate, Enddate, MONTHS_BETWEEN (Enddate, Startdate) Duration FROM
Project;
```

Next_day: finds the date of the first day of the week that is equal to or later than another specified date

```
SELECT Startdate, NEXT_DAY (Startdate, 'FRIDAY') FROM Project;
```

Round and Trunc functions also work with dates.

Round: Returns date rounded to the unit specified in the format model 'fmt'. If fmt is omitted then the date rounded to the nearest day.

Trunc: Returns date with the date truncated to the 'fmt' specified. If it is omitted then date is truncated to the nearest day.

```
SELECT ROUND (Sysdate, 'MM') FROM DUAL;
```

If today's date is June 6, 2010, the result of the above query will be today's date rounded to the nearest month. Since the date 6 is less than half to the number of days in June it is rounded to the beginning of the month. So the resultant date is 01 – Jun – 10

Similarly if the date is rounded to the nearest year

```
SELECT ROUND (Sysdate, 'YY') FROM DUAL;
```

The result of the query would be 01-JAN-10. Since the number of days till June 6 is less than half of the number of days in the year, the date is rounded to the beginning of the year.

SYSDATE: returns the system time and date and it does not take any arguments. This function also acts as a [pseudocolumn](#).

```
SELECT * FROM Product_Detail WHERE ExpiryDate < SYSDATE;
```

This query returns the details of the products that have crossed the expiry dates.

```
SELECT SYSDATE FROM DUAL;
```

Arithmetic Functions

The arithmetic operations can be done on the data using the arithmetic functions. Some of the arithmetic functions are:

- [Round](#)
- [Trunc](#)
- [Ceil](#) and [Floor](#)
- [Power](#)
- [ABS](#)
- [Mod](#)
- [SIGN](#)
- [SQRT](#)
- [Log](#)
- [Trigonometric functions](#) like – **COS, COSH, SIN, SINH, TAN, and TANH**

CEIL and FLOOR:

CEIL returns the smallest integer greater than or equal to its argument.

FLOOR does just the reverse, returning the largest integer equal to or less than its argument

```
SELECT Ceil (Price) "Apprx" FROM Product;
```

If the price of a product is Rs. 12.25, then it would return the next highest whole number; that is 13.

```
SELECT Floor (Price) "Apprx" FROM Product;
```

If the price of the product is Rs. 12.25, the whole number lower than this value is returned; that is 12.

ROUND:

ROUND (expr, n) function rounds the number to the nearest n decimal places. If n is omitted then, then it defaults to zero and it is rounded to the nearest whole number.

If n is negative then numbers to the left of decimal points are rounded.

```
SELECT Round (Price) "Approximate Price" FROM Product;
```

If the Price was Rs. 12.25, the result would be 12. If the Round function had the second argument

```
SELECT Round (Price, 1) "Approximate Price" FROM Product;
```

The result would be price rounded to one decimal place; that is 12.3.

The Round function shows a special behaviour when the second argument takes a negative value. If the second argument is having a value of -1, then it is rounded to the nearest tenth place, -2 then to the nearest 100th place, -3 then to the nearest 1000th place and so on.

```
SELECT Round (Price,-1) "Approximate Price" FROM Product;
```

The result for price Rs. 55.25 rounded to the nearest tenth place would be 60.

```
SELECT Round (Price, -2) "Approximate Price" FROM Product;
```

The result for price Rs. 155.25 rounded to the nearest 100th place would be 200.

```
SELECT Round (Price, -3) "Approximate Price" FROM Product;
```

The result for price Rs. 1555.25 rounded to the nearest 1000th place would be 5000.

TRUNC:

TRUNC (Expr, n) function truncates the Expr to n decimal places.

```
SELECT Price-Trunc(Price) "Discount" FROM Product
```

If the Price is Rs. 12.25, the result would be 12. If the Round function had the second argument

```
SELECT Trunc (Price, 1) "Approximate Price" FROM Product;
```

The result would be price truncated to one decimal place; that is 12.2.

Like Round function, the Trunc function also has the special behaviour when the second argument takes a negative value.

If the second argument is having a value of -1, then it is truncated to the nearest tenth place, -2 then to the nearest 100th place, -3 then to the nearest 1000th place and so on.

```
SELECT Trunc (Price, -1) "Approximate Price" FROM Product;
```

The result for price Rs. 55.25 truncated to the nearest tenth place would be 50.

```
SELECT Trunc (Price, -2) "Approximate Price" FROM Product;
```

The result for price Rs. 155.25 truncated to the nearest 100th place would be 100.

```
SELECT Trunc (Price, -3) "Approximate Price" FROM Product;
```

The result for price Rs. 1552.25 truncated to the nearest 1000th place would be 1000.

ABS: Returns the absolute value of the number given as argument.

```
SELECT ABS (A) ABSOLUTE_VALUE FROM Numbers;
```

If the value of A is -12, then the function returns the absolute value of 12.

MOD: function gives the modulus

```
SELECT MOD (100, 2) FROM Dual;
```

Here it returns the value zero (0) since the remainder when 100 is divided by two is zero.

EXP: Enables you to raise *e* to a power

```
SELECT A, EXP (A) FROM Numbers;
```

If the value of A is 10, this query returns the value as e^{10}

POWER: To raise one number to the power of another

```
SELECT A, B, POWER (A, B) FROM Numbers;
```

If the value of A is 3 and B is 2, the Power function returns the value 9, which is 3^2 .

```
SELECT Power (Price, 2) "New Price" FROM Product;
```

SIGN: returns -1 if its argument is less than 0, 0 if its argument is equal to 0, and 1 if its argument is greater than 0

SQRT: The function SQRT returns the square root of an argument

LN and LOG: returns the natural logarithm of its argument

COS, COSH, SIN, SINH, TAN, and TANH: provide support for various trigonometric concepts. They all work on the assumption that n is in radians

```
SELECT A, COS (A) FROM Numbers;
```

Character Functions

The character functions are used to perform character operations on character strings like concatenation of strings, getting a substring, changing the case etc there are character functions etc.

Case Manipulation Functions:

LOWER and UPPER: Used to change the case of characters to Lower case or Upper case.

```
SELECT FirstName, UPPER (FirstName), LOWER (FirstName) FROM Employee;
```

If the FirstName is Lily, the result of the query would be

```
Lily      LILY      lily
```

INITCAP: Capitalizes the first letter of a word and makes all other characters lowercase

```
SELECT INITCAP (FirstName) FROM Employee;
```

If the FirstName is 'LILY' the query would return 'Lily'

Now let us take a look at the **Character Manipulation functions:**

LPAD and RPAD: functions are used to pad a character string on the left side or right side with a specific number of characters. LPAD and RPAD take a minimum of 2 and a maximum of 3 arguments. The first, is the character string to be operated on, second the number of characters to pad it with, and the optional third argument is the character to pad it with

```
SELECT Firstname, LPAD (Firstname, 10,'*') FROM Characters;
```

If the firstName is Lily, the result of the query would be

```
*****Lily"
```

Here we need to notice that the character string 'Lily' is not padded with 10 '*' symbols, but it would be padded with '*' till the total length of the character string becomes 10. In this case it would be 6 '*'s

If the third optional argument is left out, then the string is padded with blank spaces.

REPLACE: function is used to replace a substring of a function with another set of characters. It has three arguments; the first is the string to be searched, second is the search key and third is the optional replacement string.

If the third is left out or is NULL, each occurrence of the search key on the string to be searched is removed and is not replaced with anything.

```
SELECT REPLACE (Firstname, 'ly','lian') FROM Characters;
```

If the FirstName is "Lily" then the result would be "Lilian".

TRANSLATE: as the name suggests is used to translate a set of characters to another. It takes the three arguments – the string to be searched, the set of characters to be searched and the set of characters to replace it with.

```
SELECT TRANSLATE (Firstname, 'ly','ti') FROM Characters;
```

Though the translate function looks similar to [REPLACE](#) function, the working is different. Here any occurrence of 'l' – the first character of the second argument- in the FirstName is replaced with the first letter in the third argument – 't'. Similarly, 'y' is replaced with 'i'.

If the FirstName has a value 'Lily', the result of the above query would be 'Liti'.

LTRIM and RTRIM: These functions are used to remove a trailing character or set of characters from the left or right end. LTRIM and RTRIM take one or two arguments. The first argument is a character string. The optional second element is either a character or character string or defaults to a blank.

```
SELECT RTRIM (Firstname, 'ly') FROM Characters;
```

If the FirstName is "Lily" the result would be "Li".

```
SELECT LTRIM (RTRIM (Firstname, 'ly')) FROM Characters;
```

This query would simply remove any blank spaces on the left and right side of the FirstName.

TRIM: function trims the heading or trailing characters (or both) from a character string.

TRIM (Location) FROM Dept;
 If the Location is "AMERICA" then the result would be "AMERICA". If the Location was "America" then the result would be "America".

SUBSTR: This function enables you to take a piece out of a target string

SELECT Firstname, SUBSTR (Firstname, 2, 3) FROM Employee;
 If the firstName was "Lilly", Substr ('Lilly', 2, 3) -> ill
 Substr ('Lilly', 2) -> illy

INSTR: The Instr function helps to find out where in a string a particular pattern occurs.

SELECT Firstname, INSTR (Firstname, 'O', 2, 1) FROM Characters;

CONCAT: The concat function is equivalent of || operator.

SELECT **CONCAT** (Firstname, Lastname) "FULL NAMES" FROM Employee;

LENGTH: The length function returns the length of its lone character argument

SELECT Firstname, LENGTH (RTRIM (Firstname)) FROM Characters;

CHR: Returns the character equivalent of the number it uses as the argument

SELECT Code, **CHR** (Code) FROM Characters;

Conversion functions

As in any other language, while programming we may need to convert data in one format to another. The conversion functions are used to convert the data from one form to another. Three of the popular conversion functions used in Oracle is:

- To_Char
- To_Number
- To_Date

In addition to these three Oracle 9i has other conversion functions like TO_NCHAR (), TO_CLOB (), TO_NCLOB (), UNISTR (), ASCIISTR (), ROWIDTONCHAR () and CHARTOROWID (). The discussion on these conversion functions is outside the scope of this course.

TO_CHAR

This function is used to convert the argument provided to the to character string.

The To_char function works with both numbers and dates.

To_Char with dates:

To convert date in default format (DD-MON-YY) to any format we use the To_Char function with dates. It takes two arguments, the first is the date and second is the format to which to convert the date.

For example if we want today's date to be displayed in the format "June13, 2010" then we can write the query as:

SELECT To_Char(sysdate, 'Month DD, YYYY') FROM Dual;

Here we can see the different formats that can be used in the To_Char function:

The result when used with the sample date 10-Jun-2010, Tuesday 10:30:24 PM is also given.

Format	Description	Result
Mon	The current month abbreviated.	Jun
Day	The current day of the week.	Tuesday
MM	The number of the current month.	06
YY	The last two numbers of the current year.	10
DD	The current day of the month.	10
YYYY	The current year.	2010

DDD	The current day of the year since January 1.	161
HH	The current hour of the day.	10
Mi	The current minute of the hour.	30
Ss	The current seconds of the minute.	24
a.m.	Displays a.m. or p.m.	PM
HH24	Displays the hour in the 24 hour notation	22

To_CHAR With numbers:

To_Char function is used with numbers to convert the number provided as argument to the specified format.

```
SELECT To_Char(Price, '99,999') FROM Product;
```

If the price is Rs. 10000, it will be displayed as '10,000'.

The other formats that can be used with numbers are:

	Description	Eg:	Result
9	Numeric position. (No: of 9s display the width)	9999	1234
0	Display leading zero	09999	01234
\$	Floating dollar sign	\$9999	\$1234
.	Decimal point in position specified	9999.99	1234.00
,	Comma in position specified	999,999	1,234
MI	Minus sign to the right	MI9999	1234-
PR	Parentthesize negative numbers	PR9999	<1234>
EEEE	Scientific notation	99.999EEEE	1.234E+03
V	Multiply by 10 times	9999V99	123400
B	Display zero values as blank and not zero	B9999	

TO_NUMBER

The to_number function is used to convert a character string into a number.

```
SELECT To_Number (data) FROM Characters;
```

If data has the value "10" then the result would be numeric 10.

TO_DATE

The To_Date function is used to convert the date specified in any format to the default date format. It takes two arguments the first argument is the date to be converted and the second argument is the format in which the first argument is given. For specifying the formats, the format notations that we saw for the To_Char function can be used.

If we get the date value from an application in the format DD-MM-RR, then should be converted to default format before it can be inserted to the table.

```
INSERT INTO Emp (Empid, Hiredate)
VALUES (101, TO_DATE ('10-10-05','DD-MM-RR'));
```

Now let us see some miscellaneous functions that cannot be put under any of the previous categories:

GREATEST and LEAST

The greatest and least are used to find the largest and smallest values from a set of values. Though it looks similar to the aggregate functions Max and Min, Greatest and Least are not aggregate functions. They are used to work on a set of values, which are explicitly given as arguments, or a set of values of a single row. Both these functions can take any number of arguments as different from the Max and Min functions that take only one argument, which is the name of the column.

```
SELECT GREATEST ('ALPHA', 'BRAVO', 'FOXTROT', 'DELTA') FROM DUAL;
```

USER

The function User returns the name of the current user of the database. It also acts as a [pseudocolumn](#).

```
SELECT USER FROM DUAL;
```

If the user had logged in as “Scott” the result of the above query would be Scott.

Having seen all the commonly used functions in Oracle, Let us now see some **functions that work with NULL**: These functions work with NULL and can take any data types as arguments

NVL (expr1, expr2):

This function returns expr2 if expr1 is null, else it returns expr2.

Consider that we need to find the total salary that is the sum of salary and commission. Assume that the commission is an optional field. That is it can take NULL values. We know that any arithmetic operation involving NULL would result in NULL. But this is not what we need. Substituting NULL with zero can solve this problem.

So the query that does the arithmetic operation can be written as:

```
SELECT ename, sal, comm, sal + NVL (comm, 0) FROM Emp;
```

NVL2 (expr1, expr2, expr3):

This function returns expr2 if expr1 is not null it, else it returns expr3.

NULLIF (expr1, expr2):

Nullif returns NULL if both parameters are equal in value otherwise it returns 1.

```
SELECT NULLIF (1, 1) FROM Dual;
```

This query would return NULL since both the arguments of Nullif are equal.

COALESCE (expr1, expr2, ..., exprn):

The Coalesce function returns the first non-NULL value in the argument list.

```
SELECT COALESCE (NULL, NULL, '3') FROM Dual;
```

The output of this query is 3.

In oracle we have some functions that give us the flexibility of checking for conditions. They are Decode and Case.

DECODE

The decode function has the functionality of an IF-THEN-ELSE statement.

The general syntax is:

DECODE (expression, search, result [, search, result]... [, default])

Where, **expression:** is the value to compare.

Search: is the value that is compared against **expression**.

Result: is the value returned, if **expression** is equal to **search**.

Let us understand this with an example. I have given indentation at the right places so that the code is more readable though these indentations are not needed for the successful processing of the query.

```
SELECT supplier_id,
       DECODE (supplier_id,
               10000, 'IBM',
               10001, 'Microsoft',
               10002, 'Hewlett Packard',
               'Gateway') SUPPLIER_NAME
FROM suppliers;
```

How does the query work?

Here, any occurrence of the value 10000 in the supplier_id column would be substituted by 'IBM'; 10001 by 'Microsoft' and 10002 with Hewlett Packard. Any value other than 10000, 100001 and 10002 would be substituted with 'Gateway' in the output.

CASE:

The case statement as you can see in the sample code is a more flexible extension of DECODE statement. It can do value comparisons and also condition check and it is more readable than the DECODE function.

```
SELECT Productname , Comm_prct,
       (CASE Comm_prct
          WHEN 0.1 THEN 'Low'
          WHEN 0.15 THEN 'Average'
          WHEN 0.2 THEN 'High'
          ELSE 'NA'
        END) Commission
FROM Product
ORDER BY Commission;
```

Here any occurrence of 0.1 in the comm_prct column would be substituted with 'Low', 0.15 with 'Average' 0.2 with 'High' and any other value with not applicable ('NA').

Conditions check can also in the case statement.

```
SELECT Productname , Comm_prct,
       (CASE
          WHEN (Comm_prct >=0.1 AND Comm_prct< 0.15) THEN 'Low'
          WHEN (Comm_prct >=0.15 AND Comm_prct< 0.15) THEN 'Average'
          WHEN (Comm_prct >= 0.2 AND Comm_prct<0.3) THEN 'High'
          ELSE 'Na'
        END) Commission
FROM Product ORDER BY Productname;
```

GROUP BY Clause

“The GROUP BY Clause is used to group the data according to the value of an attribute and then apply aggregate functions on each of these groups”.

Consider that the customer wants to find the cheapest product in each description.

Try to find the cheapest using your logic. How was it worked out?

1. Take the Products of a description together. OR Group the Products according to the description
2. Find the cheapest in each group.

The database will also use the same logic to find the answer.

```
SELECT Min (Price) FROM Product GROUP BY ProductDescription;
```

The features of the GROUP BY clause are:

- It groups data together based on the value of a column.
- Aggregate functions are used to compute aggregate operations on the data grouped by GROUP BY clause
- Only the column used in the GROUP BY clause can be used as such in the SELECT clause. All the other columns should use some aggregate function.

HAVING

The having is used when we need to add a condition that uses an aggregate function. This is because the WHERE Clause has a restriction that Aggregate functions cannot be used in the WHERE Clause.

The HAVING clause should always be given after the GROUP BY Clause.

Suppose the problem that we want to solve is ‘Customer wants to find the cheapest product in each description and he needs the details only if the cheapest item is costlier than Rs. 100

Here we have to Group the data based on the Description and apply the aggregate function Min () in the filtering condition.

```
SELECT Min (Price) FROM Product GROUP BY ProductDescription HAVING Min (Price)>100;
```

The HAVING clause is used when we have to use a filter condition that includes an aggregate function.

SUB QUERY

Before we discuss on [what is a subquery](#), let us see why we need a subquery.

What would we do if we need to find something from a table based on the dynamic result of a query? For

Example; if you need to find the list of product having the same price as Closeup Cool crystals; how would you do that?

1. Find the price of close up cool crystals
2. Find the products having that price

If a query needs to work based on the output of another query, then we use the subquery.

ie; the value feeding the WHERE clause is dynamic, then subquery needs to be used

SQL can also be written in the same logic

```
SELECT ProductName
      WHERE Price= (SELECT Price FROM Product
                    WHERE ProductName= 'Close Up Cool Crystals');
```

This way of nesting SQL queries inside another SQL query is called **Nested Queries** or **Sub Queries**. Here the inner query works and based on the result of the inner query the outer query works.

The **different types of subqueries** are:

1. [Single row subquery](#)
2. [Multiple row subquery](#)
3. [Multiple column subquery](#)
4. [Scalar subquery](#)
5. and [Correlated subquery](#)

The subqueries can be classified according to the values returned by the query, as:

- Single row subquery.
 - Multiple row subqueries
- Multiple column subqueries and scalar subquery.

Single Row Subquery:

The Single Row subquery returns single row.

```
SELECT ProductName WHERE Price=(SELECT Price FROM Product WHERE ProductName= 'Close Up Cool Crystals');
```

Here the equality operator is used since the inner query returns only one row.

Multiple Row SubQuery:

The Multiple row subquery returns more than one row.

```
SELECT ProductName FROM Product WHERE Price IN (SELECT Price FROM Product WHERE
ProductDescription = 'ToothCare');
```

Notice the difference in the operator used in the WHERE clause for comparing values in the outer query with the values returned by the inner query.

Here the IN operator is used, since the inner query returns multiple values and to compare a value with a set of values we use the IN operator.

If we have to check whether a value is greater than or less than a set of values, we can use the following operators.

- >ALL – gives the effect of more than Max in the set to which value is compared.
- <ALL – gives the effect of less than the Min in the set to which value is compared.
- Any – has the same effect of IN
- >ANY- gives the effect of more than the Min in the set to which value is compared.
- <ANY – gives the effect of less than the Min in the set to which value is compared.

Multiple Column Subquery:

The multiple column subquery returns multiple columns of a single or multiple rows.

```
SELECT ProductName FROM Product WHERE (ProductDescription,Price)=SELECT  
ProductDescription,Price FROM Product WHERE ProductName='Colgate ActiveSalt';
```

Here the equality operator is used to compare the values of Outer and inner queries since the inner query returns only a single row.

Correlated SubQuery

A correlated subquery is a nested query in which the inner query is dependent on the value retrieved by the outer query and outer query is dependent on the value returned by the inner query and hence the inner query cannot complete its work first. So both these queries work simultaneously; row by row.

Consider the query

```
SELECT Name FROM Students St WHERE St.studid = (SELECT Nc.studid FROM Ncc Nc WHERE Nc.studid  
= St.studid);
```

The query returns the details of students who are also part of NCC. This type of query is called the Correlated subquery. Here the inner query and the outer query work simultaneously.

Most of the correlated subqueries can be rewritten using the [JOIN](#)

Scalar SubQuery:

The subqueries that return a single value is called scalar subquery. It can be used in SELECT statements and in DML statements.

```
INSERT INTO Max_credit_Tb (Name, Max_credit)  
VALUES ('Bill', SELECT Max (credit) FROM Credit_table WHERE name = 'BILL');
```

Let us also see some special subqueries.

Inline views

Though the [inline views](#) are discussed under the category of views, they are basically subqueries in the FROM clause.

Subquery in the SELECT clause:

Till now, we saw subqueries in the WHERE clause and FROM clause. Subqueries can also be used in the SELECT clause.

```
SELECT Name, (SELECT Sum (pay) FROM Repay WHERE PayID= b.Payid) FROM Payee b;
```

What would be the output of this query?

This query would return the name of the payee and the amount that payee has repaid.

TCL

Let us first understand [what a transaction is?](#)

What is the logical set of activities when money has to be transferred from one account to another?

1. Reduce amount from account 1
2. Add the amount to account 2

After the successful transaction, the change should be made permanent (Saved)

If one of the steps in the transaction fails then the whole transaction should roll back.

So we need a concept which enables the working of a set of DML statements as a single unit. It should save the changes if all the DMLs worked and rollback if at least one DML in the set failed

So a **transaction** is a logical set of changes. A database transaction consists of:

- Any number of DML statements that can be treated as a single entry or logical piece of work.
- Or one DDL Statement; since auto-commit works with DDL statement.

A transaction should follow the [ACID properties](#) of a transaction.

The Transaction manager, Buffer Manager, Lock Manager and Recovery Manager takes care of the different aspects of a transaction in the DBMS.

A database transaction begins with the first DML statement encountered and ends with:

- A [COMMIT](#) or a [ROLLBACK](#) is issued
- A DDL or DCL statement executes
- The system crashes.
- The user session ends; that is, the user logs out

So the TCL is **Transaction Control Language** which is used to manage the transactions in the database. It consists of two commands Commit and Rollback.

The **COMMIT** statement is used to save the data manipulations till the commit statement. Automatic commit works for DDL statement.

The **ROLL BACK** statement is used to discard all the changes till the previous commit. An automatic rollback occurs at system failure.

Now let us see some of the **features of TCL**.

- Auto-commit works with DDL; that is; after a DDL statement is executed, the changes prior to the statement is automatically saved.
- Once commit is given the saved changes cannot be rolled back.
- Automatic commit occurs at log out.

Oracle uses a **SAVEPOINT** to mark the important points of transaction. It is similar to a bookmark. Using Savepoint we can mark certain points up to which we may need to rollback.

For example we issue the following statements- a COMMIT statement, an INSERT statement and a DELETE statement. A SAVEPOINT named t1 is set after the INSERT statement. If we give a simple ROLLBACK statement, then all the effects of the statement till the COMMIT statement will be rolled back. But we need to rollback only the DELETE statement.

For this we can tell the database to ROLLBACK till the SAVEPOINT by using the syntax ROLLBACK TO t1. Here only the effect of the statements till SAVEPOINT t1 would be rolled back.

```
COMMIT;  
INSERT INTO test VALUES (1,'test');  
SAVEPOINT t1;  
DELETE FROM test WHERE col1=1;  
ROLLBACK TO t1;
```

Concurrency Control

In a database more than one transaction can occur concurrently. If the DBMS does not take adequate measures, these concurrent transactions would lead to inconsistency of data.
Let us analyse **Why concurrency control is needed**.

If concurrent transactions are allowed in an uncontrolled manner, it would lead to unexpected results. Some of the problems are:

1. **Lost update problem:** Consider that one transaction, X writes a value of a datum (data-item) over the value written by a concurrent transaction, Y. Thus the first value written by the transaction Y is lost even though there may be other concurrent transactions which need to read this value.
2. **The temporary read problem (uncommitted dependency):** This problem happens when a transaction X reads a value written by a transaction, Y that is later aborted. The value written by the transaction Y is lost on abort, but the transaction X has already read the value written by the Y before it was aborted. This causes the transaction X to proceed with inconsistent data.
3. **The incorrect summary problem (Incorrect Analysis):** A transaction X takes a summary over values of a repeated data-item. In the mean time a second transaction, Y updates some instances of that data-item. The resulting summary produces a random result, depending on the timing of the updates, and whether a certain update result has been included in the summary or not.

Concurrency control in DBMS ensures that concurrent transactions are performed without violating the integrity constraints of the database. It is one of the important [functions of the database](#). To maintain the correctness of the transactions, the DBMS ensures that the consistent state of the database is recoverable after a failed transaction. The DBMS guarantees that no 'committed' transactions are lost and effect of aborted transactions are do not remain in the database. That is the transaction maintains the [atomicity](#). A transaction that meets the [ACID rules](#) ensures that the DBMS has implemented concurrency control.

[Atomicity](#), [Consistency](#) and [Isolation](#) is achieved through concurrency control.

Concurrency control mechanisms

The concurrency control mechanisms are categorised as

Optimistic: In this the transactions are blocked or aborted only if a problem is encountered. Otherwise no locks are maintained.

Pessimistic: Locks are maintained for all transactions irrespective of whether there is a scope for problems. The different methods of concurrency control come under one of these categories or are a combination of both.

Some of the methods of concurrency control are:

Two-Phase Lock Protocol: Here the transaction is completed in two phases- Expansion and contraction. In expansion phase, the [locks](#) on the different rows are acquired. In the contraction phase the locks are released and no new locks are acquired.

MVCC: Here a database will implement updates by marking the old data as obsolete and adding the newer version. So there are multiple versions stored of which only one is the latest. MVCC provides "point in time" consistent views. Read transactions under MVCC use a timestamp or transaction ID to determine what state of the database to read. Thus it avoids managing locks for read transactions because writes can be isolated because of the old versions being maintained, rather than through a process of [locks](#). MVCC provides each user connected to the database with a "snapshot" of the database for that user to work with. Any changes made by the user will not be seen by other users of the database until the transaction has been committed. MVCC is supported by Oracle, MySQL-InnoDB, MySQL-Falcon, PostgreSQL

Locks

Locks are one of the mechanisms used for concurrency control in a DBMS. It prevents a transaction from accessing a data that is being modified by another transaction. There are two types of locks.

Exclusive locks: A transaction that gains an exclusive lock will be able to read and modify the data item.

Shared Locks: A transaction that gains a shared lock will be able to read the data but will not be able to modify it.

The locks are gained implicitly. But exclusive locks can also be gained explicitly for a data item read for modification by using the SELECT FOR UPDATE statement.

The locks work in the following way:

- A transaction that needs to access a data item first locks the item. It requests for a shared lock or exclusive lock.
- The lock will be granted if this data item is not locked by any other transaction.
- If the data item is locked, the DBMS determines whether the current request is compatible with the existing lock. If a shared lock is requested on the data item that already has shared lock on it, request will be granted. Otherwise transaction must wait until the existing lock is released.
- Transaction holds a lock until it is explicitly released, either during execution or when it terminates with [rollback](#) or [commit](#). Only when the exclusive lock is released the effects of the write operation will be made visible to another transaction.

Though locks are one of the methods to achieve concurrency of transactions, it may also lead to [dead locks](#).

Dead Lock

Deadlock is a situation in which two or more transactions are waiting for one another to release the locks.

For example, Transaction X might hold a lock on some rows in the *Product* table and needs to update some rows in the *Order* table to complete the transaction. In the mean time, transaction Y holds locks on those very rows in the *Order* table but needs to update the rows in the *Accounts* table held by Transaction X. So both transactions are not able to complete the transactions due to the locks held by them. All activity comes to a halt until the DBMS detects the deadlock and aborts one of the transactions.

ACID Rule for database transactions

The ACID rule for transactions is a set of properties that ensure that database transactions are processed reliably. The properties were defined by Jim Gray in 1970, but the acronym was coined by Andreas Reuter and Theo Haerder.

ACID stands for Atomicity, Consistency, Isolation and Durability.

Let us see each of these.

Atomicity: requires that a transaction has to be processed in its entirety or not at all. That is if a part of the transaction fails the entire transaction should fail. It is important that the database should maintain the atomic nature in spite of any failure.

The database failure can occur due to Hardware failure (eg: hard disk failure), System failure (eg: network failure), database failure (eg: lack of space to store data) or application failure (eg: application posts wrong data).

Consistency:

The consistency requires that the database should always be in a consistent state after every transaction. The rule does not state how the consistency of data should be maintained. This is left to the implementers of the database.

For example the consistency of records of a parent table and child table are maintained by referential integrity rule. When a row is removed from the parent table, the database can choose to deal with it three ways.

- Abort this transaction and rollback to the previous consistent state.
- Remove the rows in the child table referring to the row in the parent table
- Nullify the corresponding fields in the child rows that point to the deleted row of the parent table.

Different DBMS choose one these as the default behaviour. Some DBMS also let the user chose the proffered behaviour.

Isolation:

The property of Isolation requires that transactions that are occurring concurrently should be isolated from each other. That is one transaction should not interfere in the work of another. The only exception is that one transaction may be made to wait in the queue while another transaction is being processed.

If the transactions are not isolated, it may lead to inconsistency in the data. While one transaction is modifying a row, another may read the uncommitted data and modify it. This would lead to one of the transactions to fail and thus lead to inconsistency of data.

Durability:

Durability requires that the DBMS should be able to recover all the committed transactions in the database.

Once a transaction is committed, it is a guarantee to the user that this data can be recovered. To enable this many of the databases maintain a transaction log to write the transactions. In the case of any system failure, the transactions can be recovered from this transaction log. For this reason a transaction is considered to be committed only after it is written to the transaction log. In Oracle the transaction log is the [Redo Log file](#).

Architecture of the database

Knowledge of the internal working of the database is important to be a good database programmer and DBA. The understanding of physical structure and the logical structure of the database also becomes important to DBAs.

All databases will have the following parts

Database Manager: This is the interface between the database and the application that interacts with the database

File Manager: that manages the allocation of space on the disk and the data structures representing the information on the disk.

Query processor: that translates the queries in the English like language to low level instructions database manager would understand.

DML Precompiler: translates the DML queries to low-level instructions. That is; DML precompiler will convert the [Non-procedural DML](#) that the programmer writes to [procedural DML](#)

DDL Compiler: Translates DDL statements to the form that the database manager understands. It also stores the details of the objects created in the data dictionary.

Architecture of Oracle Database

To understand the architecture of the database, we need to understand the Physical Layer and the Logical Layer of the Oracle database.

The Physical Layer

Oracle database contains the following 3 physical files that conform to the Operating System in which the database works.

Data Files:

Oracle database contains one or more data files that physically store the data of the logical database structure, which includes the tables and indexes. The data files have the extension .ora. Each .ora file will have 1 MB size. The database would add more data files according to the amount of data that has to be stored in the database. These data files can be associated with only one database.

Redo log files:

Oracle database has two or more redo log files that hold all information for recovery in case of system failure. It records all the changes made to the database information. In case of abnormal shutdown, the changes that have been made to the database up to that time can be obtained from the redo log.

The redo log file can be used to recover a database to the state it was in before crash. The process of applying a redo log during recovery is called **rolling forward**.

A simple explanation of this process is like this. The different commands that are recorded in the redo log file from the time of creation of the database are taken and executed till the command just before the crash of the database. Thus the database is in a way recreated to the state just before the crash.

Control files:

There would be one or more control files that record the physical structure of database. It contains the database name, names and locations of the data files and Redo files, time stamp of the database creation and parameters required to start up the database.

Apart from these three files the Oracle database also uses files that are not part of database like:

- Parameter files** that stores a list of initialization parameters and value for each parameter. It contains the overall instance configuration, such as how much memory should be allocated to the instance, the file locations etc.

- Password file** that are used to authenticate the DBA. This is because if the authentication detail is stored in the database, Oracle cannot access the database before the instance is created. Hence it is stored in a password file outside the database.
- Archived redo log file** that are offline copies of redo log files.

Having seen the Physical Layer of the Oracle database, let us glance through the

Logical Layers of the Oracle database

Logically the database is divided into table spaces and schemas.

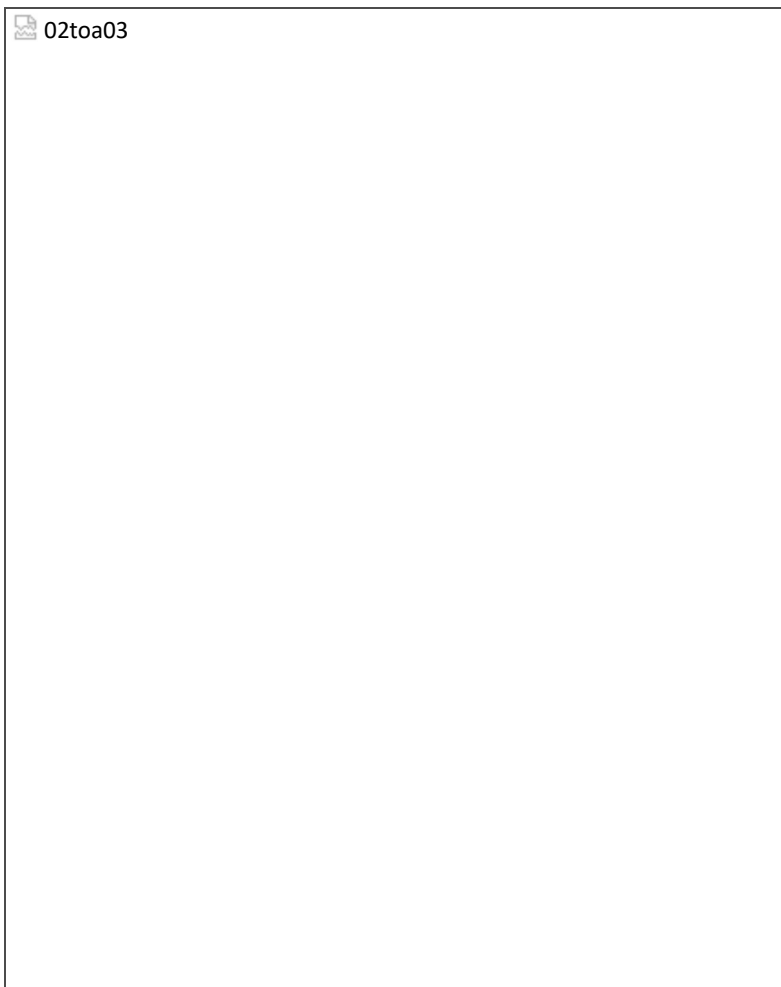
The **table spaces** are used to group related logical structures together for efficient performance. For example the objects for a major application/ branch of a company can be stored in a table space. Each table space may be mapped to one or more data files

Schema is a collection of related objects belonging to a user that defines how the user sees the database data. Schema objects are logical structures that directly refer to the database's data. Schema name will be same as the user name.

In Oracle the space used for data storage is controlled by logical structures that are the **units of database space allocation**:

- Data Blocks**: are the lowest units of space allocation in the database. Size of the data block depends on the operating system.
- Extents**: consists of specific number of contiguous data blocks.
- Segments**: are set of extents that follow a logical structure. Extents may or may not be contiguous. In the Oracle database, the first set of contiguous blocks, set up automatically when a segment is created, is called the **initial extent**. After the initial extent has been filled, the program allocates more extents automatically. These are known as **next extents**. The total number of extents that can be allocated in a database is limited by the amount of storage space available.

Any discussion on the Architecture is incomplete without a mention of the **Oracle Memory Structure**



Oracle uses shared memory for different purposes. This shared memory is divided into different memory structures. The basic memory structures are SGA and PGA

System Global Area (SGA) is the portion of the memory is allocated when the database is started on the server. SGA consists of:

Database Buffer: which is a cache in the SGA and used to hold the data blocks read from the data files. The performance of the database depends upon the size of the database buffer. The bigger the database buffer, the higher the performance. Oracle manages the space available in the database using the least recently used algorithm.

Redo log buffer: Contains changes in the data blocks in the database buffer. When the redo log buffer is filled, the log writer process writes the information to the log files. This is also used during system recovery.

Shared Pool: is the part of the SGA used by all users. It consists of the dictionary cache that is the information from the data dictionary and the library cache that is information about the most recently issued SQL command. Both of these caches help to improve the performance of the database. If a table is queried multiple times each time the database does not have to check whether the table exists since it would be stored in the dictionary cache. Likewise if the same SQL query is given more than once, it need not be parsed again since the parse tree and the execution plan will be there in the library cache.

Oracle also has some special memory pools.

Large pool: Whenever other processes need memory they take it from shared pool. This can be prevented if the large pool is configured. This is an optional memory component and if configured, the extra memory

needed for processes like Input-output would be taken from the Large pool and not the shared pool. It holds items for

Java pool is used when java application connects with the Oracle database. It handles the memory for Java methods and classes.

The PGA is a memory area that contains data and control information for the Oracle server processes. In Oracle there are two types of processes – User Processes and Oracle processes.

The User process is the user's connection to the database. It manipulates the users input and communicates with the Oracle server process through the Oracle program interface.

The Oracle process performs functions for the users and is split into Server process and background process. Server processes, also known as shadow processes communicate with the user and interact with Oracle to carry out the user's requests.

Background processes are used to perform various tasks within the RDBMS system. Following are brief descriptions of the nine Oracle background processes:

- **DBWR (Database Writer)**-DBWR writes dirty data blocks from the database block buffers to disk. When a transaction changes data in a data block, that data block may not be immediately written to disk. The DBWR usually writes only when the database block buffers are needed for data to be read. Data is written in a least recently used fashion.
- **LGWR (Log Writer)**-The LGWR process writes data from the log buffer to the redo log.
- **CKPT (Checkpoint)**-The CKPT process signals the DBWR process to perform a checkpoint and to update all the data files and control files for the database to indicate the most recent checkpoint. A checkpoint is an event in which all modified database buffers are written to the data files by the DBWR. This process is optional. In its absence, the LGWR assumes these responsibilities.
- **PMON (Process Monitor)**-PMON keeps track of database processes and cleaning up if a process prematurely dies (PMON cleans up the cache and frees resources that might still be allocated). PMON is also responsible for restarting any dispatcher processes that might have failed.
- **SMON (System Monitor)**-SMON performs instance recovery at instance start-up. This includes cleaning temporary segments and recovering transactions that have died because of a system crash. The SMON also defragments the database by coalescing free extents within the database.
- **RECO (Recovery)**-RECO cleans transactions that were pending in a distributed database. RECO is responsible for committing or rolling back the local portion of the disputed transactions.
- **ARCH (Archiver)**-ARCH is responsible for copying the online redo log files to archival storage when they become full. ARCH is active only when the RDBMS is operated in ARCHIVELOG mode. When a system is not operated in ARCHIVELOG mode, it might not be possible to recover after a system failure. It is possible to run in NOARCHIVELOG mode under certain circumstances, but typically should operate in ARCHIVELOG mode.
- **LCKn (Parallel Server Lock)**-Up to 10 LCK processes are used for inter-instance locking when the Oracle Parallel Server option is used.
- **Dnnn (Dispatcher)**-When the Multithreaded Server option is used, at least one Dispatcher process is used for every communications protocol in use. The Dispatcher process is responsible for routing requests from the user processes to available shared server processes and back.

Non Procedural DML

The DML that deals with 'What' to do and does not mention anything about 'How' to achieve what it wants, is called the Non Procedural DML. The DML that database programmer and we use are Non Procedural DML since it mentions only what to do. The Non Procedural DML is written in English like language so that we can easily build queries with it.

Procedural DML

The DML that also includes 'How' to achieve what it wants is called Procedural DML. The database manager understands only Procedural DML. It includes the 'What to do' and also 'How to achieve it' parts.

Data Dictionary

The data dictionary is a special part of the database that contains the details of all the objects in the database. It contains the metadata. The inbuilt objects like the user_tables, user_constraints, user_views etc. that contain the details of tables, constraints created and the views in the database are part of the data dictionary. The data dictionary is in the System tablespace.

DUAL

Dual is a dummy table that is provided by Oracle. It is useful to do experiments and calculations especially when there is no suitable table in the database to do this. It is provided to complete the SELECT statements syntax by giving it a FROM clause.

Features of the Dual table are:

- It has one single column named Dummy and a single data 'X'
- You can select anything from the Dual table.
- You would not be able to do DML operations on the Dual table.

Example:

If we need to see today's date, we can write the query like this.

```
SELECT Sysdate FROM Dual;
```

To do a calculation:

```
SELECT 5000*20.12 FROM Dual;
```

Pseudo Column

Pseudo columns are special columns that are not part of any table but can be used with any tables. The pseudo columns are an inbuilt feature of the oracle database. It is used in the same context as the oracle database column, but unlike the database column some of the pseudo columns are not saved in the disk.

Some of the pseudo columns available in the oracle database are:

1. [ROWNUM](#)
2. [ROWID](#)
3. [CURRVAL](#)
4. [NEXTVAL](#)
5. [SYSDATE](#)
6. [SYSTIMESTAMP](#)
7. [UID](#)
8. [USER](#)
9. [LEVEL](#)
10. [ORA_ROWSCN](#)

Some of these pseudo columns are used in scenarios that you may not need now. But it is good to have awareness of these pseudocolumns. So let us see each of these pseudo columns briefly.

ROWNUM

The ROWNUM is a pseudo column which gives row numbers to all the rows retrieved by the select query. For example we may need to see serial numbers of the rows along with the data that we retrieve from the table. There is no serial number stored in the table but we can solve this by using the pseudo column ROWNUM.

```
SELECT ROWNUM "Serial Number", ProductName FROM Product;
```

The result would be:

Serial Number	ProductName
1	CP
2	Pesdt
3	CIUp

.....

.....

Here we can see that the Product table does not have a column named ROWNUM but we are able to retrieve it from the Product table.

ROWID

The RowID is the pseudocolumn that indicates the mapping of the physical location of a row in a data file. This is mainly used for internal purposes of the database and should not be relied upon by the programmer as a constant value over a period of time.

At times this can be used to remove the duplicate rows in a table. But it is rarely used.

CURRVAL and NEXTVAL

These two pseudo columns are used with the Sequences, which are objects used to generate sequential numbers.

NEXTVAL generates the next value of a sequence. CURRVAL returns the current value of the sequence. It does not generate a value. The CURRVAL can be given only after the NEXTVAL is given only once.

If we want to know what the last value generated by the sequence named ProdID_Sequence, we can use this query.

```
SELECT ProdID_Sequence.CURRVAL FROM Dual;
```

This would work with any table in the database and not just dual. But the current value of the sequence will be retrieved as many times as the number of rows in the table used in the FROM clause.

SYSDATE

This pseudo column gives the current data and time.

```
SELECT SYSDATE FROM Dual;
```

SYSTIMESTAMP

This pseudo column gives the time stamp corresponding to the current data and time. The time stamp is similar to date type, but it can store time element up to fractional seconds.

UID

The UID returns the unique ID of the current user who is logged in.

```
SELECT UID FROM DUAL;
```

USER

The UID returns the schema name of the current user who is logged in.

For example if the current session belongs to the user 'Scott';

```
SELECT User FROM DUAL;
```

The SELECT query would return 'Scott';

LEVEL

The LEVEL pseudo column is used with the SELECT CONNECT BY to retrieve hierarchical data in a tree form from a table. LEVEL returns the LEVEL number of the node in the tree structure starting with 0 for the root level.

ORA_ROWSCN

The ORA_ROWSCN returns the conservative upper bound system change number (SCN) of the most recent change to the row, for each row. This pseudocolumn is useful for determining approximately when a row was last updated. The ORA_ROWSCN would return values corresponding to each row of a table only if the table was created with rowdependencies. Otherwise the ORA_ROWSCN would return the same value for all the rows in a [data block](#).

Inline Views

Inline views are nothing but SELECT queries that are written in the FROM Clause just as if the query was a table name. Here the data returned by the SELECT query acts as the virtual table from which data is retrieved. It is often used to simplify complex queries.

Consider that we need the salary and the names of all the employees who has the highest salary in each department.

We **Cannot** write a query like this:

```
SELECT Name, DepartmentId, Max (salary) maxsalary
FROM Employee
GROUP BY DepartmentId;
```

Since only the columns used in the GROUP BY clause can be used as such in the SELECT clause.

This can be solved by using inline views like this:

```
SELECT a.Name, a.salary, a. DepartmentId, b.maxsal
FROM employees a,(SELECT DepartmentId, max(salary)maxsalary
FROM Employee
GROUP BY DepartmentId) b
WHERE a. DepartmentId = b. DepartmentId AND a.salary < b.maxsalary;
```

Virtual Column

Virtual columns are columns that are derived from the columns in the table by applying arithmetic operations or functions on these columns.

Take the case of this query:

```
SELECT ProductName, Price FROM Product;
```

In this query all the columns in the SELECT clause are original columns of the Product table.

But look at this query:

```
SELECT Upper (ProductName) , Price-Price * 0.1 FROM Product;
```

Here the columns used in the SELECT columns either use a function or an arithmetic operation and hence are virtual columns.

Whenever we use a virtual column it is a good practice to use a column alias since it not only makes the query more readable, but it also makes the output more meaningful. If you do not give the [column alias](#) you would get column headings like 'Upper (ProductName)' and 'Price-Price * 0.1' which is not very good.

Let us rewrite the query:

```
SELECT Upper (ProductName) "Product Name", Price-Price * 0.1 "Discounted price"  
FROM Product;
```

Performance Tuning

Before we look into [What is performance tuning](#), let us see some of the problems that we faced while writing the queries.

This would give us an insight into why we also need to know the order in which the clauses in the retrieval queries are processed.

While we wrote queries many of us would have faced questions like these:

Why the column alias works in ORDER BY and not in the WHERE clause?

Where do we give the filter condition – in WHERE, HAVING or ON?

Why some queries are faster than the other even though they solve the same problem?

Looking deep into the order in which the different clauses in the SELECT statement are parsed will help in the answering the above questions.

Many applications that are developed in the stock market, railways etc. Many a times, the application takes time to create reports. The applications that are data intensive will take time, in report creation and data retrieval, if the query is not properly written. This results in wastage of user's time.

This becomes a problem since the time factor is important in the business domain and hence the need for performance tuning.

Performance tuning is the process of improving the performance of the SQL query or a piece of code in database programming by adopting the right methods and conventions.

The **different tips that can be used in tuning the performance** are:

- Take advantage of the [order in which the queries are parsed](#)
- [Use indexes where ever applicable](#)
- [Re-look at the design of the database](#)
- [Use stored procedures](#)
- [Use bulk select where ever applicable](#)
- Finally, [hardware upgrade](#)

We will see each of these points in some detail. Leaving out any of these points from discussion would make the discussion incomplete. Even if you are not familiar with some of the points mentioned, you would be able to appreciate its importance once you read it.

To fully understand how to implement queries that gives high performance we need to know the **Order in which the clauses are parsed**:

The list below gives the order in which the different clauses in the SELECT statement are parsed. As we can see, the order parsing is **different from** the order in which the clauses appear in the SELECT statement.

FROM
WHERE
GROUP BY
HAVING
SELECT
DISTINCT
ORDER BY

Each clause makes a virtual table and passes it to the next clause. So the important point in performance tuning is to filter data at the earliest possible level to improve the performance of the query. Let us go into some more detail and understand something more about what happens for each clause.

The FROM Clause:

This clause works first in the SQL statement. The work happens in the following sequence.

- Whole data is selected from the table that is in the FROM clause. If there is a JOIN all the data is selected from the two tables concerned and a Cartesian product is made.
- This virtual table containing the Cartesian product is passed to the ON clause. Any filter added here is applied on the virtual table containing the Cartesian product and the next virtual table with related data is formed.

- If the JOIN contains more than two tables, mention the table with the lower number of rows in the left side. This is because the first two tables are joined together first, the ON condition is applied and then the third table is joined to it. Mentioning the tables with lower number of rows first will reduce the size of the virtual table that is passed to the next stage.
- The extra columns needed for outer joins are added at this stage.

The WHERE clause:

In the WHERE clause, mention the condition that is more likely to be evaluated to false first. This would ensure that the other conditions need not be evaluated. So when it comes to huge amount of data it would save time.

The SELECT Clause:

The columns are filtered from the virtual table, made while parsing the FROM clause, at this level. The column alias is formed at this stage. This is the reason why column alias does not work in the WHERE clause. The vertical filter has not been done and the column alias has not been assigned to when the WHERE works, but it will work in the ORDER BY clause since this clause works after SELECT clause.

Important points for performance tuning:

- In the JOIN having more than two tables, mention the table having the lower number of rows first
- Filter the rows at the earliest possible level in this order – JOIN, WHERE and then HAVING. Pass the filter to the next clause only if the filter cannot be done at the current filter level.
- Whenever possible filter in the ON clause.
- Filter in the WHERE than HAVING where ever possible.

2. Use of Indexes

We use the concept of Indexes to improve the performance of the SELECT queries. Indexes improve performance in the similar way as to how an index of a book helps in making searching of a topic faster.

The following criteria can be used to choose the columns for indexing.

1. The columns frequently used in the WHERE clause
2. The columns frequently used for sorting in the ORDER BY clause
3. The columns frequently used in the GROUP BY Clause
4. The columns involved in JOIN

The indexing will improve the speed of the queries

1. If the table is not frequently updated.
2. If there are high number of unique values
3. The number of rows to be returned is low. There is no exact rule which states what percentage of rows this indicate. But generally it is taken as upto 25% of the rows.

Indexing will affect performance negatively if it is used in the following scenarios

1. If the column returns a high percentage of the column rows. In this case the performance will be higher without the index, since there will be loss of time due to the frequent switch in reading between the table and the index. Here whole table scan will be faster than the query which uses indexes
2. For the same reason index on column with low number of unique values will hinder performance (Eg: a Gender field)
3. Tables that are frequently updated should not be indexed since there is an additional load of maintaining the indexes for changes made to the data.
4. If the column contains a large number of NULL values
5. If the wild card used is very generic, it will return large number of rows. So the advantage of indexes are not taken
6. Too many indexes will reduce performance

3. Design of the Database

If the performance of the queries did not improve even after the tuning of the queries and the re-look at the indexes we can look at the design of the database. We can do the following:

1. Check whether the database is [normalized](#).
2. If the normalization results in a lot of joins, then some [de-normalization](#) can be done to improve the performance of the queries.

4. Use stored procedures wherever applicable

Stored procedures improve the performance since they take the advantage of the buffer cache. It also reduces the traffic between the application and the database. This is because stored procedures can process multiple queries and it gives the result. All this is done by the database server itself and hence there is no need for multiple interactions with the external application.

5. Use bulk select wherever applicable

Use the bulk select into [composite type](#) variables in stored procedures. It works much faster than normal iteration to retrieve from the cursor.

Eg:

```
DECLARE
    CURSOR major_polluters IS
        SELECT name, mileage
            FROM cars_and_trucks
            WHERE vehicle_type IN ('SUV', 'PICKUP');
    Names          name_varray; //declaration of variable of Varray type
    Mileages       number_varray;
BEGIN
    OPEN major_polluters;
    FETCH major_polluters BULK COLLECT INTO names, mileages;

    ... now we can work with data that has been retrieved in to the arrays ...
END;
```

6. Hardware upgrade

After applying all the possible techniques for performance tuning by working on the query and the code, if the queries are still slow, the database administrator can look into upgrading the hardware. But this should be the last option.

Normalization

Why Normalisation

In the introduction to the database, we saw that the person who maintained the data organized his data in different registers. Why did he do so? Did he get any advantages in doing this?

- It helped in keeping related info together.
- The person knew which file to look for finding a data
- The efficiency of search increases
- The scope for repeating data unnecessarily is reduced.

These points are used in the design of the tables in the database also. When it is used in database table design, it is called normalization.

Normalisation is the process of efficiently organizing data in a database so that the database is free of unnecessary redundancy and anomalies -insert, update and delete anomalies- that lead to loss of data integrity.

There are two goals of the normalization process:

- Eliminate redundant data (for example, storing the same data in more than one table) and
- Ensure data dependencies make sense (only storing related data in a table).

Both of these goals reduce the amount of space a database consumes and ensure that data is logically stored.

Now let us see the different **levels of normalisation**.

There are three main levels of normalization –

- First Normal Form
- Second Normal Form and
- Third Normal Form

In addition to the three levels of Normalisation we have the

- Fourth Normal form
- Fifth Normal form and
- Sixth Normal Form

There is also another level of normalisation called BCNF which is an improvisation on the Third normal form.

The first three normal forms were proposed by Dr. E. F. Codd who also proposed the Codd's Rules that determine whether a database is truly relational.

To understand more about the different normal forms, let us come back to the example of the store.

First Normal Form

If the all the data is in a single table there would be insert, update and delete anomalies. There would also be huge amount of redundancy in the table.

How do we ensure that the tables that we found are in the first normal form?

To ensure that the tables are in first normal form, we need to

- Group the raw data heads into related groups
- Check whether the data is atomic
- Assign a primary key to each table.

After this we assume that the tables are in the First normal Form.

Let us understand this through the example of the store. Suppose the database designer identified the main entities and attributes as follows:

- Product – Product ID #, Name, Price, Weight, CompanyCode
- Stock – ProductID #, Quantity, Reorder level, VendorCode, VendorName, Vendor Address
- Order- (Product ID, OrderDate) #,ProductName, Product Description, Item Quantity, VendorCode.

Second Normal Form:

The second normal form is applicable to tables that have composite primary keys.

Take a look at the Order table.

Product ID	OrderDate	ProductName	Product description	ItemQuantity	VendorCode
11293	12-Jan-08	Meswak	Aurvedic toothpaste	200	112
11293	13-Jan-08	Meswak	Aurvedic toothpaste	200	112
11295	15-Dec-08	Lays	Salt snack	100	113

Do you notice any disadvantages?

We can see that:

- The product name is repeating
- The Product description is repeating

Due to these disadvantages, there should be slight modification of the structure of the table. So we can consider the application of the second normal form rule.

How do we ensure that a table is in the second normal form?

The second normal form is applicable to tables that have composite primary keys. So for tables that have simple primary keys we assume that it is in the second normal form.

In the Order table, the Primary keys are the combination of ProductID and the OrderDate

The rule of the Second normal form is

“Check whether the attributes should be functionally dependant on the whole primary key. If any attribute is not dependent on the whole primary key, then separate it to another table.”

Examine the Order table and find whether there is a violation of this rule.

We can see that the Product name and description is dependent on the ProductId and not the primary key as a whole.

So these have to be separated to another table. I.e. we need to create a new table with Product id, product name and description. But since we already have a table with basic product details, it makes sense that we add these attributes to that table. So the structure of the table is modified like this.

Product table

Product ID	Name	Product description	Price	Weight gms	Company Code
11290	Colgate Active Salt	ToothCare	45	200	112232
11291	Closeup Cool crystals	ToothCare	48	200	112237
11292	Pepsodent	ToothCare	40	200	112237
11293	Meswak	Aurvedic toothpaste	30	200	112236
11294	Mavila	Aurvedic toothpaste	35	200	112235
11295	Lays	Salt snack	10	100	112233

Order table

Product ID	OrderDate	ItemQuantity	VendorCode
11293	12-Jan-08	200	112
11293	12-Jan-08	200	112
11295	15-Dec-08	100	113

Third normal Form:

Again see the structure of the tables. Do you notice any disadvantages?

1. Suppose we delete the detail of a stock, there is high probability that the details of the vendor

would be lost if he has supplied a product only once.

2. If the Name of the Vendor is to be changed, it has to be changed in all the records where it is mentioned

So there are some defects in the structure that need to be modified. Hence there is scope for one more level of normalization, the third level.

How do we ensure that the tables are in the third normal form?

In the **third level of normalization** we check for [transitive dependencies](#) and the rule is "The attributes should be functionally dependant on the Primary key and only the Primary Key. If any attributes have any dependencies other than to the primary key they should be removed to a separate table"

Do you notice any violations of this rule?

1. In the stock table the Vendor name and Address are functionally dependant on the Vendor Code

This is a violation of the third rule of normalization and these attributes are removed to another table.

So the new structure of the tables is:

1. Product – Product ID, Name, Price, Weight, CompanyCode
2. Stock – Product, Quantity, Reorder level, VendorCode
3. Vendor - VendorCode, VendorName, Vendor Address
4. Order- Product ID, OrderDate, Item Quantity, VendorCode,

The tables are now in the third level of normalization and we can notice that after this

1. The data is organized in a more meaningful manner
2. Unnecessary repetition of data has been removed

We saw the three main levels of normalisation. But we also know that there are two more levels up to which we may go. **To what level do we normalise our tables?**

Normally normalization is done up to three levels. This will ensure that the goals of normalization are met for most of the cases.

But in some rare cases where we need to take some business logic also into consideration in addition to the dependencies, we may have to go to the 4th and the 5th levels.

In a paper presented by Margret Wu notes that in a study of forty organizational databases, over 20% contained one or more tables that violated 4NF while meeting all lower normal forms. So, it is good to check for '4NF' also, to make sure that the database is free from unnecessary redundancy.

So let us briefly discuss the 4NF, 5NF, 6NF and BCNF.

Fourth Normal Form

Let us take a look at the relation which is in 3NF and see if there are any defects or anomalies.

The relation given here represents the categories and products of companies. Here we have two sets of data; Company's product categories and the Companies Retail Centres. This table is in the third normal form since the attributes are functionally dependent only on the primary key

RINo (#)	Company	CategoryName	Retail centres
1	Colgate Palmolive	ToothCare	Xy
2	Colgate Palmolive	ToothCare	AB
3	Dhathri	Skin Care	XY
4	Dhathri	Skin Care	AB
5	Hindustan Lever	Health Care	Xy
6	Hindustan Lever	Health Care	AB
7	Hindustan Lever	Skin Care	Xy
8	Hindustan Lever	Skin Care	AB
9	Hindustan Lever	Hair Care	XY
10	PepsiCo	Food	Xy
11	PepsiCo	Food	AB

12	PepsiCo	Beverages	AB
----	---------	-----------	----

What do we notice in the data?

Here we have the dependencies like retail stores have a multi-valued dependency on company name and the category name also have a multi-valued dependency on the company name. But there is no direct relationship between retail stores and category name.

So, for each additional retail centre name, the details of the company name and the category name has to repeat. So there is redundancy of data in the table. This would also lead to update and delete anomalies, hence, the need for the fourth normal form.

How do we ensure that the tables are in the fourth normal form?

This normal form was introduced by Ronald Fagin. The second, third and BCNF dealt with [functional dependency](#). But fourth normal form deals with [Multi valued dependencies](#).

The fourth normal form rule is that there should not be more than one multi valued dependency in a relation. If any attributes violate these rules then it should be separated in to another relation.

Here, in this relation, there are two multi valued dependencies.

- between the company name and Category name and
- Between the company name and the retail centre name.

This is a violation of the fourth normal form rule. So the relation has to be split up into two relations.

Company	CategoryName
Colgate Palmolive	ToothCare
Dhathri	Skin Care
Hindustan Lever	Health Care
Hindustan Lever	Skin Care
Hindustan Lever	Hair Care
PepsiCo	Food
PepsiCo	Beverages
Company	Retail centres
Colgate Palmolive	Xy
Colgate Palmolive	AB
Dhathri	XY
Dhathri	AB
Hindustan Lever	Xy
Hindustan Lever	AB
PepsiCo	Xy
PepsiCo	AB

Now the two relations are in fourth normal form

Fifth Normal form

The Fifth normal form deals with [JOIN dependencies](#).

A relation is said to be in fifth normal form if it cannot be further subdivided without loss of data.

In other words, a relation R which is decomposed to n relations R1, R2,...Rn; cannot be reconstructed back to relation R without loss of data using a JOIN; if it is

in the fifth normal form.

Sixth normal form

This was introduced by Chris Date, Hugh Darwen, and Nikos Lorentzos in 2002.

Any table is said to be in 6NF if it has only non trivial [Join dependencies](#).

This is applicable to tables that have temporal data or interval data. For example if a table has data of vendors and we want to add temporal data regarding when the data within the relation is valid. But there is a constraint that the different attributes of the relation may vary during different times and this variation is independent of each other.

The 6NF is currently used in data warehousing.

BCNF Normal form

Why do we need the BCNF normal form?

Consider that the student database has a relation with details of elective and guides. Each candidate has a

guide who determines the elective.

This table is in third normal form. The candidate key is Student code and elective since a student can have more than one guides who gives different electives.

The [functional dependencies](#) are:

StudentCode, Elective-> Guide

Guide->Elective.

This relation has a drawback that if a student 38 is removed from the relation the data that the data that Mr. Martin can guide Chemistry is removed. Similarly a data that a new guide can guide Physics cannot be recorded unless a student is assigned to the guide. Due to these drawbacks, we go for the BCNF.

How do we ensure that a table is in BCNF?

StudentCode	Elective	Guide
13	Chemistry	Mark
13	Commerce	Mary
38	Chemistry	Martin
78	Physics	Majnu
77	Chemistry	Mark

The BCNF or Boyce-Codd Normal Form deals with [candidate Keys](#) that overlap. The rule for BCNF is that all determinants should be a candidate keys. If any of the determinants are not candidate keys, it should be separated to another table. It was proposed by E.F. Codd and Raymond F. Boyce.

The determinants in the relation are (StudentCode, Elective) and Guide. Of this Guide is not a candidate key and hence is separated to another relation.

StudentCode	Elective
13	Chemistry
13	Commerce
38	Chemistry
78	Physics
77	Chemistry

Guide	Elective
Mark	Chemistry
Mary	Commerce
Martin	Chemistry
Majnu	Physics

Now the relations are in BCNF.

The different types of Keys

Key – is any attribute that is used to identify a tuple or a set of tuples in a relation.

Super Key- is a key used to uniquely identify a tuple in a relation.

Candidate Key- is a super key for which no subset of it is a super key.

Primary Key – is the candidate key chosen by the database designer to uniquely identify a tuple in a relation.

Types of dependencies

Dependencies are relationships between the attributes of a table. There are different types of dependencies.

Functional Dependency relates one value of A to one value of B. I.e; for a relation having two attributes X and Y; $R(X,Y,Z)$; X and Y are said to have a functional dependency if there are zero or one value of Y for every value of X.

The relationship between the ProductID and ProductName in the Product table is a functional dependency.

Full Functional Dependency

An attribute X is said to have full functional dependency on Y if it has dependency only on Y

Transitive Functional Dependency

Is a functional dependency in which in a relation $R(X, Y, Z)$, $X \rightarrow Y$ and $Y \rightarrow Z$. In other words, transitive dependencies would mean transitive dependencies as in the mathematical or logical reasoning. For Example, the logic like if 'a' is greater than 'b' and 'b' is greater than 'c'. We can conclude that 'a' is greater than 'c' describes a transitive dependency.

Multi Valued Dependency (MVD)

In a multi valued dependency, $X \twoheadrightarrow Y$ defines a relationship in which a set of values of attribute Y are determined by a single value of X. That is for a relation having three attributes $R(X, Y, Z)$, there is said to be a multi valued dependency if for a value of X there are zero or more values of Y and Y values depend on X and have no dependency on Z.

The relationship between a ProductID and the OrderDate in the Order table is a multi valued dependency since order for a product can be placed more than once on different dates.

Trivial Multi valued dependency

A Multi valued dependency is said to be trivial if the relation $R(X, Y)$ cannot be decomposed any further. In other words a MVD is trivial if Y is a subset of X or X and Y together forms the relation.

Non Trivial multi valued dependency

A MVD of a relation is said to be non trivial if it can be decomposed into smaller relations without loss of data. For a relation to have non trivial MVD it should have at least three attributes and two of which are multi valued.

Join Dependency

A relation R is subject to join dependency if the relation can be recreated after it is decomposed into two or more relations. The join dependency is said to be trivial if one of the relations in the join can reconstruct the table having all the attributes of the relation being reconstructed.

De-normalisation

Why do we need this concept?

We saw that normalisation is used to organise the data so that the data dependencies make sense. This helps in improving the performance since

- we know where to find the data
- The data in each table is smaller and more manageable. So the retrieved data is not heavy.
- Update and delete becomes simpler.

But the increase in the number of tables also means that when ever we need a summary report, we need to join tables. If the application that we are designing is large, especially related to space research experiments or big equipments, the database also would be large and complex. The number of tables in the database and the data in each would be large. In this scenario, if we want a report it may mean joining a huge number of tables. A SELECT statement with a large number of JOINS would take a big toll on performance.

How would we solve this problem?

This problem can be solved by using the process of de-normalisation.

De-normalisation is a process of improving the read performance of the database by maintaining some redundant data or by grouping the data. De-normalisation is not same as no normalisation. De-normalisation should take place only after the database design is satisfactorily normalised.

There are two ways of implementing de-normalisation.

One is to keep the logical design normalised but allow the DBMS to maintain a additional redundant data.

Here the DBMS is responsible for the consistency of the additional data. In Oracle this is done as materialised views.

The second method is to explicitly add tables to store redundant data in logical design and it is the database designer's responsibility to maintain the consistency of the data. Here additional constraints are added to maintain the consistency. Here the SELECT is made faster but at the cost of INSERT, UPDATE and DELETE. The load of the additional constraints makes the DML operations slower.

Good Practices in SQL query writing.

When it comes to programming it is not just enough that you get your output. But how you got your output, the performance of your code, maintainability of the code etc are very important. You become a good programmer only when your programming skills are combined with the good practices in programming. As in other programming SQL also has some good programming practices.

Let us get familiar with those.

1. **Follow [coding standards and conventions](#):** This is a very important rule which many of the programmers neglect, but at their risk. Coding standards are not only for readability and resultant maintainability, but in database programming, not following the recommended coding standards may result also lead to undesired behaviour of the code. This is true for PLSQL programming.
2. **Write [optimised queries](#):** Often amateur programmers blame the database server for poor performance. But making hardware changes should be the last step for improving the performance. Writing optimised queries improves the performance to a great extent.
3. **[Normalise your database](#):** Normalisation helps in improving the performance of your database. So make sure that your database design is normalised before implementing it.
4. Follow the **anti breakdown rules** while writing queries
 - a. While you write the SELECT query explicitly mention the columns that you want to retrieve than use the "SELECT *..."
 - b. While you write the INSERT statement, mention the columns to which the values are to be inserted.

Coding standards and conventions for SQL

The coding standards for SQL are very simple.

1. All keywords should be in Upper case
2. It is a good practice to write each clause of a query in a separate line.
3. When a query is written in multiple lines, give necessary indentation
4. Give constraint name to all constraints. Use meaningful names that indicate the use of the constraint.
5. Follow these rules while giving names to objects and attributes
 - a. The object name should be singular
 - b. The object name should be in Sentence case
 - c. The attribute names should be in lowercase or sentence case
 - d. The names of objects other than tables should end with object name. For example the name of a sequence should be like this ProductIDSequence.
 - e. It is good to avoid underscores and space in between the object names and the attribute names.
 - f. Give meaningful names to the object and attribute names.
6. Use [Column alias](#) when you use a [virtual column](#) or [table alias](#) with the column name.
7. Assign a primary key for all tables in the database.

The other objects in database

Till now we were discussing about the object table. There are other objects in the database. They are

Views

Indexes

Synonyms

Sequences

Why do we need Views when we already have tables?

To understand this let us take the example of an application dealing with the processes of a manufacturing company. The junior person working on this type of application development may not be proficient in database applications. But there will be a need to search and retrieve the data. Suppose he wants to take data from different tables. He may write a very large query containing joins, aggregate functions etc. It would be difficult for him to manage. The manager also may not want to expose the whole database to him.

The other scenario is,

According to the hierarchy, we can restrict the data. Consider how systems are managed in an IT company. There is one administrator who has the full power, which means that he can see all the data in the server or in the network. But there is another level of people who has partial access to the data. Other than that, we may also have the junior staffs that have the access to very specific data only and even if they have access to some data, they may not have the permission to manipulate the data.

The same scenarios, described earlier, can be implemented in databases using views.

- The programmer who interacts with the database to retrieve the data may not be very good in programming. So he may want to retrieve all data using a simple `SELECT * FROM tablename;` syntax.
- The admin may need to restrict the access to some data in the database.
- The application designer may want certain queries to have high performance.

These advantages can be got using the concept of Views.

What are views?

- Views enable you to view or present data in a different format than it appears on the disk.
- Views are subsets of one or more tables.
- View is often referred to as a virtual table since it does not have data of its own.
- Views do not take up physical space in the database as tables do.

Suppose the programmer needs the following data; the Productdescription and cheapest product in the product type. We can create a view like this.

```
CREATE VIEW MinPricedProduct(ProductDescription, MinPrice) AS SELECT ProductDescription,
Min(Price) FROM Product GROUP BY ProductDescription;
```

A view with the following structure will be created

View name: MinPricedProduct
Columns: ProductDescription, MinPrice

Data can be retrieved from the view in the same way as we query a table.

```
SELECT ProductDescription, MinPrice FROM MinPricedProduct;
```

The Definition of the View (SELECT ProductDescription, Min (Price) FROM Product GROUP BY ProductDescription) works in the background and retrieves the data FROM the Base table. Every time the view is queried, this query works in the background. So the query and its results will be stored in the database's memory, hence, the increase in performance of the query.

General syntax for creating the view

```
CREATE VIEW view_name [(alias1, alias2...)] AS subquery [WITH CHECK OPTION] [with READ ONLY];
```

With check option would impose the restriction that we can insert into the view only that which can be retrieved from the View

With Read Only makes the view read only. I.e; DML operations cannot be done through the view that is read only.

Most common restrictions you will encounter while working with the view

- You cannot use the INSERT statement unless all NOT NULL columns used in the underlying table are included in the view (key preserved tables). This restriction applies because the SQL processor does not know which values to insert into the NOT NULL columns.
- If you do insert or update records through a join view, all records that are updated must belong to the same physical table
- If you use the DISTINCT clause to create a view, you cannot update or insert records within that view
- You cannot update a virtual column (a column that is the result of an expression or function)

To Remove a view from the database use the DROP statement.

```
DROP VIEW ViewName;
```

Why do we need Indexes?

To understand the concept of index, let us try to find the answer to these questions. What is the use of the index of a book? Based on what are the keywords in the index made? The answer is that it makes the search faster. The words that most of the readers search for are chosen as

1. The indexes are made based on the keywords (words based on which the search is done).

What are indexes?

In database also, there is a concept of index and it is used to make the SELECT query faster. The index should be created on the attribute that frequently comes in the WHERE clause of the SELECT query (The

attribute based on which the search is to be done).

The Primary keys are automatically indexed since many searches are done with the Primary key attribute in the WHERE Clause.

Suppose a lot of searches are done with the Productdescription in the WHERE clause. Indexing the ProductDescription attribute would increase the speed of all the select Queries having the ProductDescription in the WHERE clause.

```
CREATE INDEX Pro_Descr_Index ON Product (ProductDescription);
```

Please note we cannot retrieve anything from the index. It is used only to make SELECT Queries faster.

It will not be beneficial, if the table is small

If columns are not often used as condition

Most queries return more than 2-4 % of rows.

If the table is updated frequently

Indexed columns are referenced as part of an expression.

Why Synonyms?

Some times we may have very big names like John Andrew Robinson. Normally what we do is, we call him by another name called John or Jo. The same thing happens in Oracle also.

What are Oracle Synonyms?

A synonym basically allows you to create a pointer to an object that exists somewhere else. You need synonyms because when you are logged into Oracle, it looks for all objects you are querying in your schema (account). If they are not there, it will give you a big fat error telling you that they do not exist. For example, assume from the ROBERT schema that we issue a query like `SELECT * FROM emp;` and the EMP table is not there, we get this error:

```
SQL> select * from emp;
```

```
select * from emp
```

```
          *
```

```
ERROR at line 1:
```

```
ORA-00942: table or view does not exist
```

And we can see why this is a problem because of the results of this query:

```
SQL> select table_name from user_tables where table_name='EMP';
```

```
no rows selected
```

Clearly there is no EMP table, which can be a problem. Well, the truth is that there is an EMP table in our database, it is just owned by a user called SCOTT as we can see in this query:

```
SQL> select owner, table_name from dba_tables where table_name='EMP';
```

```
OWNER
```

```
TABLE_NAME
```

```
-----
```

SCOTT EMP

Since we know the table is in the SCOTT schema, and assuming we have been granted SELECT privileges to that table, we can query the table this way:

```
SQL> select * from scott.emp;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
3333	R Freeman	BOSS		24-AUG-05			
7369	SMITH	CLERK	7902	24-AUG-05	800	20	
7499	ALLEN	SALESMAN	7698	24-AUG-05	1600	300	30
7521	WARD	SALESMAN	7698	24-AUG-05	1250	500	30

Notice that we added SCOTT to the beginning of the EMP table reference. This indicates, of course, that we want to query the EMP table in the SCOTT schema, and sure enough there is the table. A schema is another word for a user. While a “user” is an account you can log into, every user also has a “schema,” which is a virtual space for the user to create their own objects. So what does that make the words “Users” and “Schemas?” Synonyms, of course!

However, it would be a bit of a pain to have to always prefix all SQL calls to the EMP table with SCOTT, there must be an easier way. There is, the way is called synonyms. In the following sections we will discuss the creation and removal of synonyms.

A **synonym** is an alternative name for objects such as tables, views, sequences, stored procedures, and other database objects.

Synonyms are objects which contain a duplicate name of an existing object.

```
CREATE [PUBLIC] SYNONYM another_name FOR existing_name
```

Existing name can be name of a table or view or sequence

Public is used to grant permission to all users to access the object using the new name.

Useful in hiding the ownership details of an object.

Dropping a synonym

We can drop a synonym using drop command.

```
DROP SYNONYM synonym_name;
```

Why Synonyms?

Consider that we have a field that has values that increase in a sequential order. Then what is the way to implement it? Either we will have to maintain the last value inserted in the application or we have to check the database and find the last value inserted. Both these are not very elegant solutions.

Hence we have an object called the sequence.

What

The sequence is a numeric value generator. It is used to:

- Create a series of unique values.
- Generate primary key attribute values

- Create a sequence of values with specific increments

How

The code below creates a sequence. ([Read and understand the code](#))

```
CREATE SEQUENCE seq2
    INCREMENT BY 1
    START WITH 900
    MAXVALUE 1000;
```

```
CREATE SEQUENCE seq
    INCREMENT BY n
    [START WITH n]
    [MAXVALUE n | NOMAXVALUE]
    [MINVALUE n | NOMINVALUE]
    [CYCLE | NOCYCLE]
    [CACHE n | NOCACHE];
```

Some points to note:

- Default increment is by 1.
- A positive number in increment causes the ascending increment, negative number causes descending increment [in this case use MINVALUE].
- Default start with is MINVALUE for ascending and MAXVALUE for descending.
- The details of the sequences created in the database are found in the USER_SEQUENCES view in the data dictionary.
- NEXTVAL returns the next sequence value
- CURRVAL obtains the current sequence value

NEXTVAL should be issued before CURRVAL is used.

Evaluations

The evaluations given below would help you understand your grip over the subject.

1. DATABASE

Identify the wrong statement

- a. Oracle is a database implemented based on Relational Model
- b. MySQL is implemented based on Network model**
- c. Oracle 8i and higher are Object Relational databases
- d. None

Which of these cannot be a function of the database?

- a. Concurrency control
- b. Performance tuning
- c. Data independence
- d. None**

Which of the statements are true

- a. Foreign key attribute can be assigned as primary key also
- b. Foreign key can have duplicate values
- c. Foreign Key attribute can have null values
- d. All the above**

Which of these is the most popular relational database querying language?

- a. YQL
- b. D
- c. XQL
- d. None

Pick the odd one out based on three schema architecture

- a. Conceptual
- b. View
- c. Physical
- d. Intermediate**

2. DATA MODELS

Relational Databases represent data as

- a. Two dimensional tables**
- b. Tree structures
- c. Linked list
- d. Circular linked list

Which of these attributes can be chosen as a primary key attribute?

- a. Name
- b. Data of birth
- c. PIN number**
- d. Deposit amount

What is the relational model term representing the number of tuples in the relation?

- a. Cardinality**
- b. Degree
- c. Dependency
- d. Concurrency

Why is it that the hierarchical model cannot represent many to many relations?

- a. It represents data as records

- b. It uses tree model to represent relationship**
- c. It uses pointers
- d. It uses values within the records to establish relationship

3. NORMALIZATION

Which statement is false

- a. Second normal form is applicable to tables having composite primary keys
- b. It is possible that a table in fourth normal form is not in third normal form**
- c. Boyce Codd normal form is an extension of the third normal form.
- d. There is a concept called denormalization

Which of the normal forms is based on the join dependency?

- a. BCNF
- b. Fourth
- c. Fifth**
- d. None

Which of the following is not based on functional dependency?

- a. 1NF
- b. 2NF
- c. 3NF
- d. 4NF**

The normal form applicable to relations having composite primary keys

- a. 1NF
- b. 2NF**
- c. 3NF
- d. 4NF

Why do we go for denormalisation?

- a. To save space in the disk
- b. To remove update and delete anomalies
- c. To remove redundancy
- d. To improve performance**

4. DDL and DCL

How will you modify the custno field to number(5) from number(2) (syntax)

- a. ALTER TABLE Customer MODIFY custno number(5);**
- b. ALTER TABLE Customer ADD custno number(5);
- c. ALTER TABLE Customer MODIFY COLUMN custno number(5);
- d. None

How do you remove the DEFAULT setting for an attribute?

- a. By using the ALTER statement
- b. By using the DROP statement
- c. By using the ALTER with the modify clause
- d. None**

What will happen to the structure of the table, if you drop the table?

- a. Structure remains but data is lost
- b. Both structure and data is lost**
- c. None of the above

What will happen to the structure of the table, if you truncate the table?

- a. **Structure remains but data is lost**
- b. Both structure and data is lost
- c. None of the above

Choose the right statement

- a. **It is possible to create a table with the same structure as the original table excluding the data from the original table.**
- b. It is possible to create a table with the same structure as the original table but the data cannot be excluded. It has to be done by a separate delete statement
- c. None

5. CONSTRAINTS

Choose the right statement

- a. Constraints can be added at table level or column level
- b. Composite primary key can be added only at the table level
- c. More than one constraint can be applied to a single attribute
- d. **All the above**
- e. a and c only

What happens when we try to delete a row in the parent table which has child table references

- a. Deleting not allowed. Error message is shown
- b. Deleting allowed if ON DELETE CASCADE is mentioned with the DELETE statement
- c. **Both the above**
- d. None

Which of the statements are true?

- a. Constraints once added cannot be removed
- b. **Constraints other than NOT NULL is added using the ALTER statement with the MODIFY clause**
- c. DEFAULT setting once added to a column cannot be removed
- d. None

We need to apply a restriction on a column that it can take only a specific set of values. How do we do this?

- a. Apply the DEFAULT settings
- b. **Apply the CHECK constraint**
- c. Apply the RESTRICT constraint
- d. None

We need to remove the child rows when the parent rows are removed

- a. **Use ON DELETE CASCADE**
- b. Use ON DELETE SET NULL
- c. Both
- d. None

6. DML

Initial value of salary = 10000 and commission = 1000

UPDATE salary = 11000, commission = commission + 0.1 * salary;

Final values are

- a. salary = 11000 and commission = 2000
- b. **salary = 11000 and commission = 1100**
- c. salary = 10000 and commission = 2000
- d. salary = 10000 and commission = 1000

What is the effect of the query

DELETE Products

- a. The table is removed from the database
- b. **All the data of the Product table is removed.**
- c. Error: Wrong syntax
- d. None

What is the difference between IN and = operator

- a. Both have same effect
- b. IN used when a value is compared with a set of values
- c. = used when a value is compared with a single value
- d. **b and c**
- e. None

SELECT name from Employee where Name = 'A%';

What is the output?

- a. List of all name that begin with A
- b. Error message
- c. None of the above

SELECT ProductName from Product WHERE Price BETWEEN 500 and 1000;

Select the query which gives the same result as the above

- a. SELECT ProductName from Product WHERE Price >500 or Price <1000;
- b. SELECT ProductName from Product WHERE Price >=500 or Price <=1000;
- c. SELECT ProductName from Product WHERE Price >=500 and Price <=1000;
- d. SELECT ProductName from Product WHERE Price >500 and Price <1000;

7. JOIN

When will you use USING clause and ON clause

- a. USING is used with Equi join and ON with NonEqui join
- b. **Using is used when the attribute involved in the Join condition is having the same name and ON when the condition is explicitly specified**
- c. None

We need to take all the rows from one of the tables and corresponding rows from the other table. Which Join do we use?

- a. Inner
- b. **Outer**
- Cross

None

There is a relationship within the table. Which Join do we use to retrieve the corresponding attributes?

- a. Inner Join
- b. **Self Join**
- c. Internal Join
- d. None

SELECT Emp.FirstName 'Employee Name', mangr.FirstName 'Manager Name' FROM Employee Emp JOIN Employee Mangr ON(Emp.ManagerID=Manger.EmpID)

- a. Error
- b. **The query is right**
- c. None

We need to join two tables that have a parent child relationship. The Primary key and the foreign key columns that are involved in the referential integrity are having the same name. Which join can we use?

- a. Cross Join
- b. Anti Join
- c. **Natural Join**
- d. None

8. SUB QUERIES

The subquery that returns a single value

- a. **Scalar subquery**
- b. Correlated subquery
- c. Single row subquery
- d. None

The subquery that can be used as an alternative to equi joins

- a. Scalar subquery
- b. Multiple row subquery
- c. Correlated subquery
- d. None

What is the output

```
SELECT Name, Sal, DeptID FROM Emp WHERE sal IN (SELECT sal FROM Emp WHERE DOJ='10-JAN-98' ORDER BY sal);
```

- a. Displays details of all employees whose salary is same as the employees who joined on 10th January 1998.
- b. Error
- c. None

Identify the error

```
SELECT Name, Sal, DeptID FROM Emp WHERE sal = (SELECT MIN(Sal) FROM Emp GROUP BY deptID );
```

- a. Error in SELECT Clause
- b. **Error in comparison operator used**
- c. Error in FROM Clause
- d. Aggregate functions cannot be used in the inner queries

```
SELECT Min(Price) FROM Product GROUP BY ProductDescription HAVING Min(Price)>100 WHERE Productname IS NOT NULL;
```

- a. **No mistake**
- b. Error: in Group by clause
- c. Error: in Order of clauses
- d. None

9. FUNCTIONS

Identify the error

What is the output. Original value of salary = 12458.5678

```
SELECT ROUND(salary,-3) FROM Employee;
```

- a. Error
- b. 12458.567
- c. 12.4585678
- d. **12000**

```
SELECT * FROM Employee WHERE UPPER(Name)=UPPER('Mark');
```

- a. **Helps in a case insensitive search**

- b. Error
- c. None

Which function is used to remove space from the ends

- a. ltrim
- b. rtrim
- c. **Both**
- d. None

SELECT Max(Name) FROM Employee

- a. Error: Max cannot be used with non numeric characters
- b. **Query does not have syntax errors.**
- c. None

10. GROUP BY

SELECT Min(Price) FROM Product GROUP BY ProductDescription HAVING Min(Price)>100 WHERE Productname IS NOT NULL;

- a. No mistake
- b. Error: in Group by clause
- c. **Error: in Order of clauses**
- d. None

SELECT Team, AVG(Salary) FROM ORGCHART GROUP BY TEAM WHERE AVG(Salary) < 38000;

- a. Error in SELECT Clause
- b. Error in FROM Clause
- c. **Error in WHERE Clause**
- d. None

SELECT Min(salary) FROM Product GROUP BY DepartmentNo

- a. Returns 1 row
- b. Returns as many rows as the number of departments
- c. None

SELECT Name, DeptId FROM Employee GROUP BY DeptID

- a. **Error in SELECT Clause**
- b. Error in FROM Clause
- c. Error in WHERE Clause
- d. None

11. Transaction and TCL

Rollback rolls back to

- a. previous commit
- b. previous DML statement
- c. Previous DDL or DCL
- d. None

Which statement is false?

- a. Lock is a concurrency control method
- b. Auto-commit works with DDL
- c. **Auto-commit works with DML**
- d. None

Pick the odd one

- a. Atomicity
- b. Concurrency

c. Isolation

d. Definition

Consider the pseudocode

1. begin the transaction
2. insert into the employee table
3. set a savepoint spt1
4. delete an employee Antony from the table
5. Create a table Products
6. roll back to savepoint spt1
7. end the transaction

What is the final effect?

- a. delete and insert are rolled back
- b. delete alone is rolled back
- c. delete and create are rolled back
- d. No change occurs even though the rollback works**