Database Systems { i) by CJ Date ii) Candly and Baig

INTRO:

Shared collection of logically related data (and a description of this data), designed to meet the information needs of an organization.

System catalog (meta data) provides description of data to enable program-data indepedence.

Logically related data comprises entities, attributes, and relationships on organization's information.

Database is a place where data is technically stored and if we need we can connect our database whith any type of program and we can access the data anywhere from that database.

File-based system:

When there was no data base techniques, there was used file-based system.

This is collection of application program that performs services for the end users (eg reports).

Each program defines and manages its own data.

In File-based system one particular file is made for one particular program and can not be connected or accessed by

other programs.

For eg. We make a file for one program in MS, that file can not be connected or accessed by the other program that uses power point for its data files. or that program don't have MS, then you can't access that file.

Limitations / Problems of File-Based Aproach

- Separation and isolation of data
  - Each program maintain its own set of data.
  - Users of one program may be unaware of potentially useful data held by other program.

- Duplication of data
  - Same data is held by different programs.
  - Wasted space and potentially different values and/or different ~~values~~ formats for the same item.

- Data Depedency
  - Depedency on program you have in your computer.

For eg.
You made a program in java now you can't use that program in c++.

- Incompatible file Formats.
  - Programs are written in different languages, and so cannot easily access each other's file.

- Fixed Queries / Prodigeration of application programs
  - Programs are written to satisfy particular functions. (You can not alter queries of that program).
  - Any new requirement needs a new program.

Database Aproach:

- Arose because:
  - Definition of data was embaded in application programs, rather than being stored seperately and independence.
  - No control over access and manipulation of data beyond that imposed by application programs.

Result :

- the database and database management system (DBMS).

=> DBMS is a software.

In this semester we will learn

i) Oracle (oracle)

ii) mysql (apache)

other databases :

sqlserver (microsoft)

db2 (Ibm)

=> Using above software we can do database management.

Eg :

Storing / Deleting / update / change / search / sort data

=> These are relational databases

- Some important points

=> Table is also called Entity / relation

=> Row also called Tuple / record

=> Column also called attribute / fields.

Database Management System (DBMS)

- A software system that enables users to define, create, maintain and control access to the database.

- (Database) application program:
  a computer program that interacts with database by issuing an appropriate request (SQL statement) to the DBMS.

* Note that:
  Data, database and ~~dbms~~ DBMS are seperate things.

i) Data
   Raw material

ii) Database
   Collection of data stored in some particular form.

iii) Database management system
   It is a software that manage data and database.
   Eg
   Mysql, Oracle

- We can't touch database or access it. (only choosen people by DBMS to

access and see data can access that data.)
Or that data can be accessed using any language.

- Here we are studying relational data-base. Relation means table. We study:
- Tabular form database.

=> Other forms of database are
Tree Format database
Graphical database
and so on

* Database Aproach:

DDL

DML

(Definition and so on avalaible in notes).

* Views

(We will cover this after studying DDL and DML

* Component of database management system:

Hardware

Software

Data

Procedures

People

(Details avalaible in notes)

* Advantages of DBMS

- Control of data redudancy
- Data consistency
- More information from same amount of data.
- Sharing of data
- Improved data intigrity
- ... Many other, described in notes

* 3 layers of relalational DBMS
  i) External
  ii) Conceptual
  iii) Internal

* External
  - This is user view interface
  - Includes searches, interact, execute queries etc by user.
  - Maintain login details.
  - Describes that part of database that is relevent to a particular user.

* Conceptual level
  - Community view of database.
  - It is a translation b/w external and internal levels. (Conversion from english to programming codes).
  - Describes which query to use on which table.
  - Full internal information is known/accessed by conceptual but external can't access that.
  - Describe what data is stored in DB and relationships among the data.

=> Schema : Internal structure design. Backend property of table etc.

- Conceptual level doesn't know details of /abc data stored in hardware.
- Describe that part of database that is relevant to a particular user.

* Internal level
  - Low level maintainance of data.
  - Storing of data is described in this level.
  - Physical representation of the database on the computer.
  - Based on C, C++, Oracle languages.

In table column is nothing but collection of arrays.

Relationship may be:
one to one
one to many
many to many.

Structures are classes.

* Data Independence
  - Logical
  - Physical

* Logical:
  - Describes why things are in layers.
  - Because changes to one part will

not effect the others.

- Layes or parts could be changed/replaced,
- Immunity of external layers that changes in conceptual part don't effect external.

* Physical
  - Immunity of conceptual schema to changes in internal schema.
  - and so on as in logical independence.
  - more details in notes.

* SQL
  - (Structured query language).
  - Used to performs tasks on database like create, remove, changes and so on on data. in database.

Interface : Html/less

Business logic : Delete, update and so on PHP, Java, C++.

- Database doesn't understand other languages for that purpose SQL is used.

{ Interface Business database:
MVC : Model view controller

* Two parts of SQL DB language.

DDL } Already
DML } learnt

=> More explained in notes about these.

* Data Models

{ two types of DB users Admin acc SYS, SYSTEM

Network data model
Relational data model
Hierarchical data model

=> Relational data model is still successful DM.

CH # 03

The relational model transparancies

* Terminologies:

=> A RELATION is a table with columns and rows. (only applies logical structure of DB, not the physical structure).

=> RELATIONAL DATA means when data is being stored in tabular format.

=> ATTRIBUTE is a named column of relation.

=> DOMAIN is a set of allowable values for one or more attributes.

=> TUPLE is a row of a relation.

=> DEGREE is a number of attributes in a table.

=> CARDINALITY is number of tuples in a relation.

=> RELATIONAL DATABASE is collection of normalized relations with distinct relation names.

=> Cell is intersection of rows and columns.

* Relational keys

=> Primary key:
Candidate key selected to identify tuples uniquely within relation.

=> Alternate keys :

Candidate keys that are not selected to be primary key.

=> Foreign key:

Attribute, or set of attributes, within one relation that matches candidate key of some (possibly same) relation.

continoue on next page

* Alternative terminology for Relational Model.

Formal terms	Alt 1	Alt 2
Relation	Table	File
Tuple	Row	Record
Attribute	Column	Field

Database Relations

* Relational Schema

Named relation defined by a set of attributes and domain name pairs.

Backend properties.

* Relational database schema.

Set of relation schemas, each with a distinct name.

=> Super key :

An attribute or set of attributes that uniquely identifies a tuple within a relation.

=> Candidate Key

defined in slide ppt

* Properties of relation

-> Relation name is distinct from all other relations names in relational schema.

-> Each cell of relation contains exactly one atomic (single) value.

-> Each attribute has a distinct name.

-> Value of an attribute are all from the same domain.

-> Each tuple is distinct. (No duplicate tupples)

-> Order of attributes and tuples has no significance.

* Integrity Constraints

- Null
- Entity Integrity
- Referential Integrity
- General Constraints

These are described in ppt slides.

* SQL

http://localhost:8080/apex/
http://localhost:8080/apex/f?p=4950

* SQL SELECT statements.

=> Capabilities of SQL SELECT Statements.

- Projection → Selection of attributes.
- Selection → Selection of tuples.
- Join → Joining data of two tables.

Projection :

-> A SELECT statement retreives information from the database.

Eg

```
SELECT * from table.
```

OR

```
SELECT {[Distinct] column|expressio [alias],...}
    From table.
```

=> Select identifies what columns.
=> From identifies which table.
=> * means all attributes.

=> Selecting all columns.

```
Select *
From departments;
```

=> Selecting specific columns.

```
select department_id, location_id
From departments;
```

* Arithmatic expressions

+ , - , * , /(Divide)

→ An arithmatic expression can contain column names, constant numeric values, and the arithmatic operators.

→ Applies on numerical values not on string.

→ After applying any arithmatic operator the result will not be saved permanantly.

→ BDMAS Rule applies here.

→ Arithmatic operations over null returns null.

Example:

```
Select first_name, salary, Salary + 300
From employees
```

OR , ..., salary - 300

OR , ..., salary * 10

OR , ..., salary / 10

* ALIAS

Changing or renaming columns heading.

Eg

```
Select first_name AS 'First Name' Last_name
AS 'Last Name', salary
From employees;
```

→ If there is one string in AS then quotes not needed

★ Concatenation operator

- Concatenate columns or character strings to other columns.
- Represented by two vertical bars (||).

Eg

```
Select    last_name || job_id AS "Employees"
From      employees;
```

OR For space or other symbol.

```
select    last_name || ' , ' || job_id AS "Employees"
From      employees;
```

↑ This is called literal string.

=> Two strings or expressions also can be merged using a keyword 'AND'.

Command

★ /Keyword DISTINCT

It is applied to any column when we need unique values.

Eg

```
Select    Distinct department_id
From      employees;
```

★ Command DESC (DESCRIBE)

To display structure of table.

Eg

```
DESCRIBE  Employees;
```

Selection

* Limiting the rows selected
  WHERE CLAUSE

Eg

```
SELECT    *|{[Distinct] column|expression,...}
From      table
[WHERE    condition(s)];
```

→ The WHERE CLAUSE Follows FROM clause.
→ WHERE restricts the query to rows that meet a condition.
→ Condition is composed of column names, expressions, constants, and a comparison operator

* Comparison Conditions
  = , > , >= , < , <= , <> (not equal)

Eg

```
Where    salary <= 12000;
```

* Other comparison conditions
  BETWEEN ... AND ... → Between two values (inclusive)
  IN (set) → Match any of a list of values.
  LIKE → Match a character pattern.
  Is Null → Is a null value.

Eg

```
Where    salary Between 2500 AND 3500;
Where    manager_id IN (100, 101, 201);
```

→ IN condition also known as membership condition.

WHERE first_name LIKE 'S%';

% denotes many or zero characters

_ denotes one character

→ LIKE condition can be used as a shortcut for some BETWEEN comparisons.

Eg

Where hire_date LIKE '%95';

(employees who joined b/w jan 1995 and dec 1995).

→ We can combine pattern-matching characters.

Eg

WHERE Last_name LIKE '_o%';

(Names of all employees whos last names have an o as the second character).

OR '%o_

(second last o)

Null :

Where manager_id is null

(returns all rows, which have null value in manager_id)

Where manager_id IS NULL.

= ESCAPE

★ Logical Conditions

Used to merge or invert two or more than two conditions.

AND → Returns true if both component conditions are true.

Eg:

```
SELECT    employee_id, last_name, job_id, salary
FROM      employees
WHERE     salary >= 10000
AND       job_id LIKE '%MAN%';
```

OR → Returns true if either component condition is true.

Eg:

```
WHERE     salary >= 10000
OR        job_id LIKE '%MAN'
```

NOT → Returns true if the following condition is false.

Eg:

```
WHERE     job_id
          NOT IN ('IT_PROG', 'ST_CLERK');
```

(or)

Also used with between:

```
WHERE     salary NOT BETWEEN 1000 AND 15000
```

(or)

```
WHERE     last_name NOT LIKE '%A%'
```

(or)

```
WHERE     commision_pct IS NOT Null
```

* Rules of Precedence

Order Evaluated	Operator
1	Arithmatic Operator
2	Concatenation operator
3	Comparison Conditions
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] Between
6	NOT
7	AND
8	OR

→ Override rules of precedence by using parenthesis.

* 'ORDER By' Clause

→ Sort rows with the ORDER By clause.
- ASC : accending order, default
- DESC : Descending order

→ This is clause like 'select' or 'where'

→ Comes last in the select statement.

Eg

```
SELECT    last_name, job_id, hire_date
FROM      employees
ORDER BY  hire_date.
```

OR

```
ORDER BY  hire_date  DESC;   // descending order
```

=> Using two conditions in ORDER BY clause.

```
ORDER BY  Department_id , salary
          1st           2nd.
```

* Function

- Functions takes an argument input, performs an action on it and returns output.

INPUTS - Many
Output - Only one.

SQL Functions

Functions are very powerful feature of SQL and can be used to do the following.

- Performs calculations on data
- Modify individual data items
- Manipulate output for groups of rows.
- Format dates and numbers for display.
- Convert column data type

SQL functions sometimes take arguments and always return a value.

Two types of SQL Functions
- Single row Functions
- Multiple-row Functions

=> Single-Row Functions

These functions operate on single rows only and return one result per row. There are different types of single-row functions.
- Character
- Number
- Date

- Conversion
- General

Single row functions:
- Manipulate data items
- Accept arguments and return one value.
- Act on each row returned
- Return one result per row
- May modify the data-type
- Can be nested
- Accept arguments which can be a column or expression.

→ function_name [(arg1, arg2, ...)]

=> Character

Accept character input and can return both charater and number values.

→ Types of characters function
- Case-manipulation functions
  - LOWER
  - UPPER
  - INITCAP
- Character-manipulation functions
  - CONCAT → conc 2 string
  - SUBSTR → getting a part of string
  - LENGTH → Checking length of str
  - INSTR → Searching an str part
  - LPAD / RPAD → Aligning right/left

TRIM → Remove first/last letter.
REPLACE → Replacing

```
select  first_name , salary  from  employees
        LOWER (first_name) ...
        FROM  Employees
```

LOWER (M-H GAJJU) → m-h gajju
UPPER (m-h gajju) → M-H GAJJU
INITCAP (m-h gajju) → M-H Gajju

CONCAT ('Hello', 'World') → HelloWorld
SUBSTR ('HelloWorld', 1,5) → Hello
LENGTH ('HelloWorld') → 10
INSTR ('HELLOWORLD', 'W') → 6
LPAD (salary, 10, '*') → *****24000
RPAD (salary, 10, '*') → 24000*****
TRIM ('H' FROM 'HelloWorld') → elloWorld

Number
Accept numeric input and return numeric value.

- Round:
  Rounds value to specified decimal.
  Round (45.926, 2) → 45.93
- Trunc:
  Truncates value to specified decimal.
  Trunc (45.926, 2) → 45.92

- MOD

Returns remainder of division.

MOD (1600, 300) → 100

Eg

SELECT Round(45.926, 2) from dual

op 45.93

→ If we want to round left side of decimal.

round (45.923, -1)

op 50

=> Date

Operates on values of the DATE data-type.
(All date functions return a value of DATE data type except the MONTHS_BETWEEN function which returns a number).

- Oracle database stores dates in an internal numeric format: century, year, month, day hours, minutes, seconds
- Default date display format is DD-MON-RR

07-June-94.

SYSDATE is function that returns system's

- Date
- Time

SELECT Sysdate from dual

* Arithmetic with Dates

- Add or subtract a number to or from a date for a resultant date value.
- Subtract two dates to find the number of days b/w those days.
- Add hours to a date by dividing the number of hours by 24.

Eg To check weeks passed any particular time.

```
SELECT  (sysdate - hir.date)/7      // weeks
        (sysdate - hire_date)/30    // Months
        (sysdate - hire_date)/365   // Year
```

* DATE Functions

- Month_between → Number of months b/w 2 dates
- ADD_Month → Add calendar months to date
- Next_day → Next day of date specified
- Last_day → Last day of the month
- Round → Round date
- Trunc → Truncate date.

Eg

```
months_between ('6/15/2020', '1/15/2020')   → 5
Add_month ('6/19/2020', 7)                  → 1/19/2021
Next_day ('01-sep-95', 'Friday')            → 08-sep-95
Last_day ('01-Feb-95')                      → 28-Feb-95

Round (sysdate, 'Month')
        (will round and next month start if 15 days passed).
```

- Round(sysdate, 'Year')
  (Will round and will start new year if 6 month passed)
- Trunc(sysdate, 'Month')
  'Year'
  (Starts month or year from first).

→ In manually date
round(to_date('12/6/2020' 'Month')).

=> Conversion

Convert a value from one data type to other.

Two types of data types conversion
i) Implicit data type conversion
ii) Explicit data type conversion

i) Implicit

Sometimes when it needs to convert smaller to larger data type, Oracle server does it automatically. that's called implicit data-type conversion.

→ For assignments the oracle server can automatically convert the following.

From	To
Varchar2 or char	Number
varchar2 or char	Date

Number	Varchar2
Date	Varchar2

Eg

→ Char to number

```
Select * from employees
Where   Salar = '10000'
```

→ Char to date

```
Select * from employees
Wher   hire_date = '1/30/2009'
```

→ For expression evaluation, the oracle server can automatically convert the following.

From	To
Varchar2 or char	Number
Varch2 or Char	Date

Eg

```
Select '4' * '5' from dual.
```

* Explicit Data type conversion

When data type casting is larger to smaller and performed by user it's called Explicit data type conversion.

Done by using conversion functions.

→ Explicit type conversions can be done:

From	to	Functions
Char	Number	To_Number
Number	Char	To_Char
Char	Date	To_Date
Date	Char	To_char

These functions are not used in assignments or expressions. We use these functions when we perform any operation on columns.

→ To_char

To_char(date, 'format_model')

```
Select  last_name, to_char(hire_date, 'DD/MM/YY'
From    employees
```

- Format model must be enclosed in single quotation and is case sensitive.

★ Elements of the date format model.

YYYY	Full year in numbers.
Year	Year spelled out.
MM	Two digit value for month.
Month	Full name of the month.
MON	Three-letter abbreviation of the month.
DY	Three-letter abbr of the day of week
DAY	Full name of the day of week
DD	Numeric day of month

→ Time elements format the time portion of date.

HH24 :MI :SS AM = 15:45:32 PM

→ Add character strings by enclosing them in double quotation marks.

DD "of" Month = 12 of october

→ Number suffixes spell out numbers

ddspth fourteenth

Select last_name , to_char (hire_date , 'DD "of" Month YYYY')
From employees

OR DDspth (spell and th)

OR DDth (th only)

→ For time

Select to_char (sysdate, 'HH:MI:SS AM DD[MM]/YYYY)
From dual OR HH24

→ To_char with number

Perform on columns.

To_char (number, 'format_model')

→ These are some format elements you can use with the TO_CHAR function to display a number value as a character.

9	Represents a number
0	Forces a zero to be displayed.

$	Places a dollar sign
L	Uses a floating local currency symbol
.	Prints a decimal point.
,	Prints a thousand indicator.

Eg

```
Select  last_name  to_char(salary, '$00,999,999.99')
From    employees
```

* To_Number

Convert a character string to a number format.

To_Number (char [, 'format_model'])

* To_Date

Convert a character string to a date format.

To_Date (char [, 'format_model'])

* NVL Function

NVL (To_char(manager_id), 'No Manager')

=> If anywhere null value then print any specified string there.

-> Works on string only.

```
Select    last_name
          NVL (To_char (manager_id), 'No Manager')
From      employees
Where     manager_id  Is  Null;
```

* General Functions

These functions work with any data type and pertain to using nulls.

- NVL (expr1 , expr2)
- NVL2 (expr1 , expr2 , expr3)
- NULLIF (expr1 , expr2)
- COALESCE (expr1 , expr2 , ... , exprn)

=> NVL

If there is any value print that value if not then print specified char string.
Convert a null to an actual value.

-> Data-types may be date, char , and num.

-> Data-type must match:

```
NVL (commision_pct, 0)
NVL (hire_date , '01-JAN-97')
NVL ( job_id , 'No Job Yet')

NVL (manager_id , 000)
```

=> NVL2

For eg applying on salary and commision columns , if comm is null then print

salary only, in comm not null then print salary + comm.

```
Select  last_name, salary, commision_pct,
        NVL2 (commision_pct, 'SAL+comm', 'SAL') incom
From    employees;
```

=> NULLIF

Compares two expressions, If they are equal the function returns null. If they are not equal, the function returns first expression.

Eg

```
Select  last_name, length (last_name), first_name, length(
        first_name)
        Nullif (length (last_name), length (first_name)) 'Result'
```

=> COALESCE

Prints first argument if its null print 2nd arg and if its null print 3rd argument.

```
SELECT      Last_name
    COASCE (commision_pct, salary, 10) 'comm'
From    employees
```

=> Conditional expressions

Provide the use of If-then-else logic within a SQL statement.

- Use of two methods
  - Case expression
  - Decode function

- Case expression

Facilitate conditional inquiries by doing the work of an If-then-else statement.

Eg

```
Case expr When com-expr1 Then return expr1
          When - expr2 - - expr2
          -    - expr32 - - expr32
          Else else_expr]
END
```

Using the case expression

```
Select last_name, job_id, salary,
       Case job_id When 'IT_Prog' then 1.10 * salary
                   When 'ST_Clerk' then 1.15 * salary
                   When 'SA_REP' then 1.20 * salary
       Else salary END 'Revised Salary"
From employees;
```

- The decode Function.

Same as case expression.

Decode is function while case expr is not.

```
Decode (job_id, 'IT Prog', 1.10 * salary,
        'ST Clerk', 1.15 * salary,
        'SA REP', 1.20 * salary,
salary)  Revised_salary
```

End of CH#3

Database System Part (II)

CH # 9 : Creating and managing Tables

Table :

Basic unit of storage ; Composed of rows and columns.

- Tables can be created any time, even while users are using the database.
- In table size is not needed to be specified. The size is ultimately defined by the amount of space allocated to the database as whole.
- Table structure can be modified online.

* Table naming rules

- Must begin with a letter.
- Must be 1-30 characters long.
- Must contain only A-Z, a-z, 0-9, _, $ and #
- Must not duplicate the name of another object, owned by the same user.
- Must not be an oracle server reserved word.
- Names are case insensitive

* The CREATE TABLE statement.

- You must have
  - CREATE TABLE previlege
  - An storage area.

```
CREATE TABLE [schema.] table
        (column datatype [Default expr] [, ...]);
```

- Your specify
  - Table name
  - column name, col data type, col size.

* Creating Tables

- Create table

CREATE TABLE ~~dept~~ students
(rollno number(3), fname varchar(30), lname varchar(30), marks number(3));

datatyp — specify no of characters. — specify no of digits.

→ (Create and table both are reserved words).

- Watching table

Select * from students

- Checking structure

Describe students.

(Will show table schema also called table propate

- Inserting data

Insert into students values (1, 'Kami', 'Ali' 50)

SQL Command — reserved keyword — rollno — fname — lname — marks

(For 2nd student) values (2, 'MH', 'Ali' 50)

:

- leaving any column null.

insert into students (rollno, fname, lname) value (3, 'Zia', 'Ahmed')

(We left marks col null).

→ By this way we let system that which value is for which column.

- Creating primary key.

  Only write "primary key after specifying no of digits or characters of specific column while creating table.

  Eg:

  Create table courses (cours_id number(3) primary key, name varchar(30), marks numbe(3))

→ We can't leave primary key column null.

- Deleting row

  Delete From courses where course_id = 413

  (Note - that writing only 'Delete course' will delete whole table, and in database deleted data is not recoverable.)

- Updating row

  → Changing value of one cell.

  Update courses set marks = 90, name = 'Fahad'
  Where course_id = 403
  OR where course_id in (403, 404)

  ↑ SQL command

- Creating new table by copying existing table's data.

```
Create table dpt AS
select department_id, depart_ment_name from departments
where departmen_id in(10,20,30,40,50)
```

- For deleting table :

```
drop table std
```

(This is not recoverable).

★ The ALTER TABLE Statement

Use the Alter table statement to:

- Add a new column
- Modify an existing column
- Define a defalt value for the new column
- Drop a column.

Arrangement of rows and columns doesn't matter in database.

- Adding new column in existing table :

```
Alter table std add (lname varchar (30))
```

OR

```
Alter table std add (address varchar(50), phone
number (11))
```

Note :

In new column data will be added by Update Keyword not by insert Keyword.

* Deleting Column

Alter table std drop column phone

or Delete from teacher where teacher_id = 2

* Modifying datatype of existing column

Alter table std modify (address number(5))

This number only also can be changed.

* To rename the column or table

Rename dept to new_departments

or Rename employees2 to emp1

* Deleting table structure and all data

Truncate table emp

In truncate we can't use where command.

* Creating new table based on existing table

Create table employees2 As

Select Employee_ID, First_name, last_name

From employees

* Comment

Comment on table emp is 'Information of employees';

OR Comment on column teachers.t_name is "This is teachers name";

* Dictionary

=> There are two types of Delete in Oracle:

Logical	Physical
-> Doesn't delete, hide data and later deleted by user.	-> Delete data on spot physically and permenantly.
-> Take less time, less process.	-> Take more time, more process.
-> 'Set unused' keyword used.	-> Delete, Truncat, Drop

-> Both logical and physical deleted data is not recoverable.

-> Delete, Truncate, Drop already studied and mentioned in these notes.

-> Set unused

Alter table teachers set unused column hir_date.

-> Deleted permenantly later:

Alter table teachers
drop unused columns.

=> Trumcating a Table

Trucate statement

- Removes all rows from a table.
- Releases the storage space used by that table.

TRUNCATE TABLE EMP1

- Using Trucate row removal can't rolled back.
-

Comments :

Already described.

-> Viewing comments through data dictionary views.
- All_col_comments
- User_col_comments
- All_Tab_Comments
- User_Tab_comments

* Using TimeStamp datatype

Used to store both time and date in database.

Eg

Create table teachers (teacher_id number(3), t_name varchar(30), hire_date timestamp)

Insert into teachers values (1, 'Kamran', '12-jan 2002 11:45.00 PM')

* Constraints

-> Constraints means rules

-> Constraints enforce rules at the tabel level

-> Constraints prevent the deletion of a table if there are dependencies.

-> Following are constraint types

- not null
- Unique
- Primary key
- Foreign key
- CHECK

-> To check contraints

select * from user constraints

// To see constraints of particular table

Where table = 'Employees'

• Not Null

The Not Null constraint ensures that the column contains no null values. (Column without the Not Null contraint can contain null values).

eg

Create table std2 (rollno number(3), fname varchar(30) not null, lname varchar(30))

* Giving name to constraints:

-> We can give any constraint the name.

-> By giving name constraints are easy to reference

-> Names are unique.
-> We can give name to all contraints.
-> If we don't give name, they will be system named constraints. (System will give them names).
   => Example is in next constraint's (unique's) description.

. Unique

When we want to avoid duplication in any column or make any column unique.

eg:

```
Create table std1 (rollno number(3), fname
varchar(30) not null,
constraint fname_unique unique(fname))
```

. Primary Key

-> This constraints make the column primary key of table. Only one primary can be created for the table. Primary key-
   - Uniquely identify each row in a table.
-> Not nullable.

eg

```
Create table std2 (rolln number(30) primary key
fname varchar(30))
```

=> Making column Primary key by altering existing table.

Alter table std3 add constraint roll_no_pk primary key (roll no)

=> Deleting constraint

Alter table std3 drop constraint roll_no_pk

* Foreign key

-> Also called referential integrity constraint.

-> If two tables A and B are in relationship, the primary key of table A used in table B is Foreign key for table B.

-> Foreign key is duplicable.

-> Foreign key of table B must match with primary key of table A.

-> It may be null.

eg:

(Table A) Create table std5 (rollno number(3) primary key, fname varchar(30), dept_id number(3))

(Table B) Create table std6 (Fname varchar(30), Emp_id number(3) primary key, dept_id number(3) constraint dept_id_Fk foreign key (dept_id) references std5(dept_id)

→ On Delete Casecade

→ This is constraint keyword of foreign key.

→ This deletes the values of rows of foreig key in table B

→ Deletes the dependent rows in the child table when a row in parent table is deleted.

eg

constraints dept_id_fk foreign key (dept_id) references stds (dept_id) on delet cascade

→ On Deletete set null

Converts dependent foreig key value to null.

→Coding same only replace On delete casecad with on delete set null.

To show constraints:

select * from user constraint

OR

select * from user constraint where table_name = 'EMPLOYEES'

* Check

Used to add additional constraints, using coditions like if in programming.

eg

After table std7 add constraint marks_check_const check (marks >=0 and marks <=100)

(Now we can't add marks less than zero and more than 100).

→ Multiple checks can be added.

CH# 4 Displaying Data From Multiple Tables

→ Obtaining data from multiple tables

→ Using differents joinings we can obtain different columns of multiple tables but those tables should be in relation by primary and foreign keys, otherwise errors would occur.

→ In joining there will be obtained cartesian product that is performed bw joining tables.

[Cartesian product is for eg we have 2 tables with 3, 3 rows, then all rows of table A will multiply by all rows of table B. (A's 1st row × B's 1st, 2nd, 3rd rows, A's 2nd row × B's 1st, 2nd, 3rd rows, A's 3rd row × B's 1st, 2nd, 3rd rows.) (3×3 = 9 combinations will produced) Result is called cartesian product].

'where' = rows
'select' = columns.

* Oracle Properietary joins.
  - Equijoin
  - Non-equijoin
  - Outer join
  - Self join

- Equijoin

As name implies that join in which equal operator is used.

→ Alway written in where condition.

→ In equijoin rows will be selected if foreign key of first table is equal to primary key of 2nd table.

eg:

```
select first_name, salary, department_name
from employees, departments
where employees.department_id = department.department_id
        FK                      PK
```

→ We also can add other conditions here.

eg

```
and (department_name = 'Marketing'
OR
department_name = 'Sales')
```

→ If columns with same name exist in both tables then while using column, must write table_name.column.

→ To avoid writing complete name, make aliases of table names.

eg:

```
Select first_name, salary, department_name
from employees e, departments d
where e.department_id = d.department_id
```

* Joining more than two tables

```
Select first_name, department_name, city
from employeees e, departments d, location l
Where e.department_id = d.department_id
and
d.location_id = l.location_id.
```

* • Non-Equijoins

→ Joining two or more tables not necessiily joining by Primary or foreign keys.

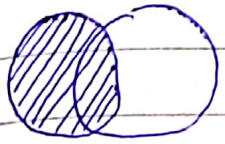
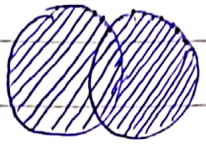
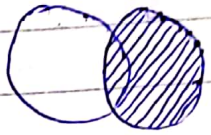
→ This joining can be done by not equal operators.

→ Non-equijoins is used when primary and foreign keys are not avalaible in both tables, and it will join tables if highest and lowest limits columns exist in one table.

eg

```
Select e.last_name, e.salary, j.grade_level
From employees e, job_grades j
Where e.salary
Between j.lowest_sal AND j.highest_sal
```

* Outer Joins

Left outer join	Full outer join	Right outer join
		
Will select rows (all) from left side table and those rows from right side which are matching with left side rows.	Will select all rows from left and right tables whether thy are matching or not	Will select all rows from ~~left~~ right side table and those rows from left side which are matching with right side rows.

→ Matching rows will be showed above and non-matching will be showed bellow.

→ These joins have 2 types of syntaxes.

- Older
- New

- Older

⇒ Left outer join

```
Select  employee_id , first_name , d.department_id ,
        department_name
From    employees e , departments d
Where   e.department_id = d.department_id  (+)
```

=> Right outer join

Syntax same only write (+) in left side in where

eg

Where e.department_id (+) = d.department_id

=> Full join was not avalaible in older way.

* New

=> Left outer join

```
Select  employee_id , first_name , d.department_id ,
        department_name
From    employees e
        LEFT OUTER JOIN
        departments d
ON      e.department_id = d.department_id
```

=> Right outer join

Syntax same , only replace 'LEFT' with 'RIGHT'

=> Full outer join

Syntax same , only replace 'LEFT' with 'FULL'