Infosys®

POWERED BY INTELLECT
DRIVEN BY VALUES

RDBMS-Day3

SQL

- Basic DDL statements
- DML statements
- Aggregate functions

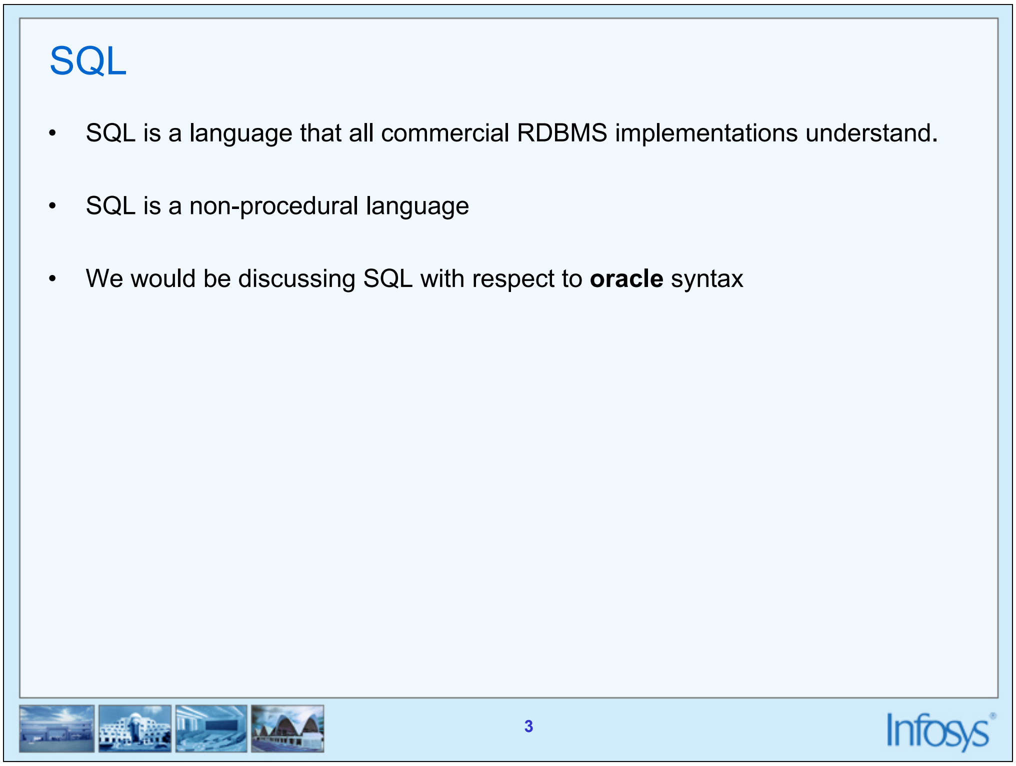
Education
and
Research

SQL

- SQL is used to make a request to retrieve data from a Database.

- The DBMS processes the SQL request, retrieves the requested data from the Database, and returns it.

- This process of requesting data from a Database and receiving back the results is called a Database Query and hence the name Structured Query Language.



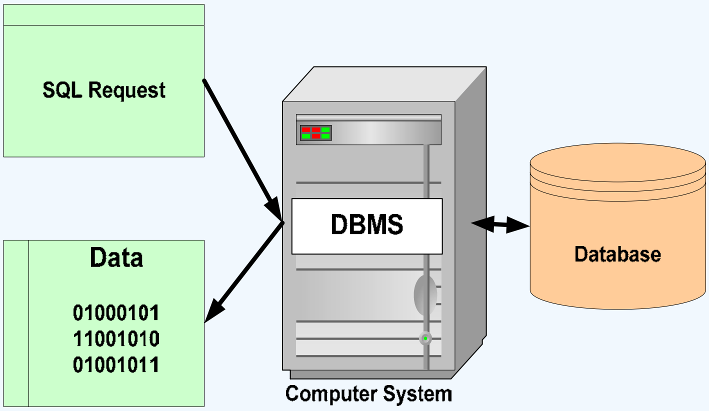
SQL

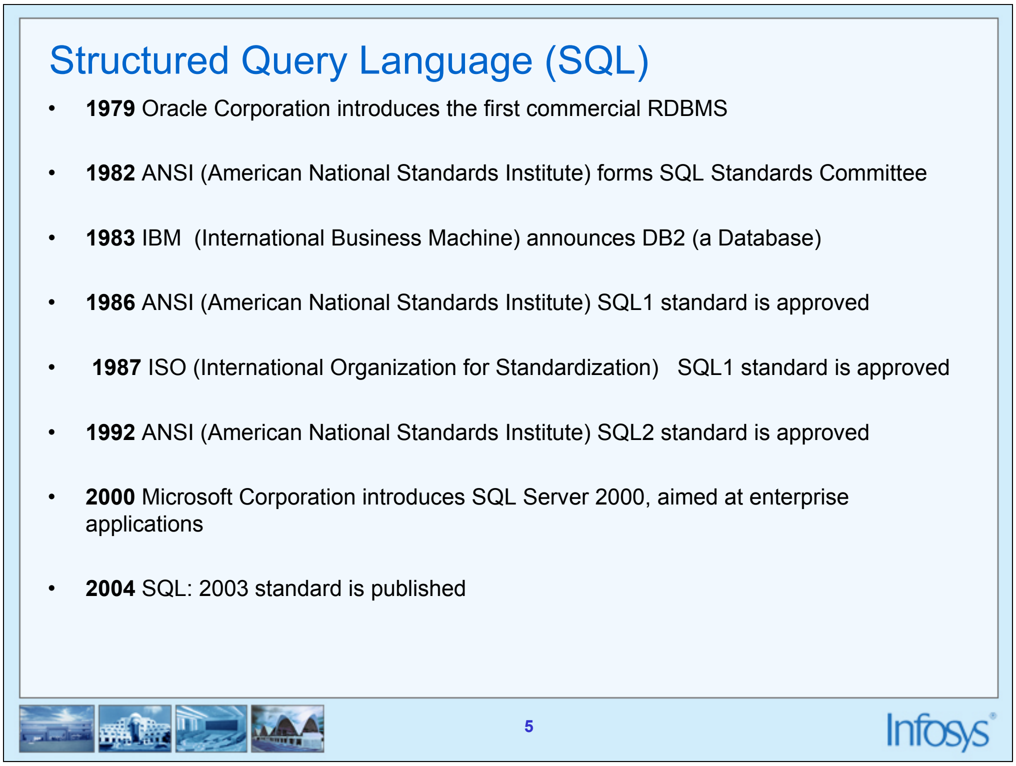
- SQL is a language that all commercial RDBMS implementations understand.
- SQL is a non-procedural language
- We would be discussing SQL with respect to **oracle** syntax

You can't write programs like the ones you would have done using C language

You can only write questions in English like language called queries which will fetch some data rows from the database.

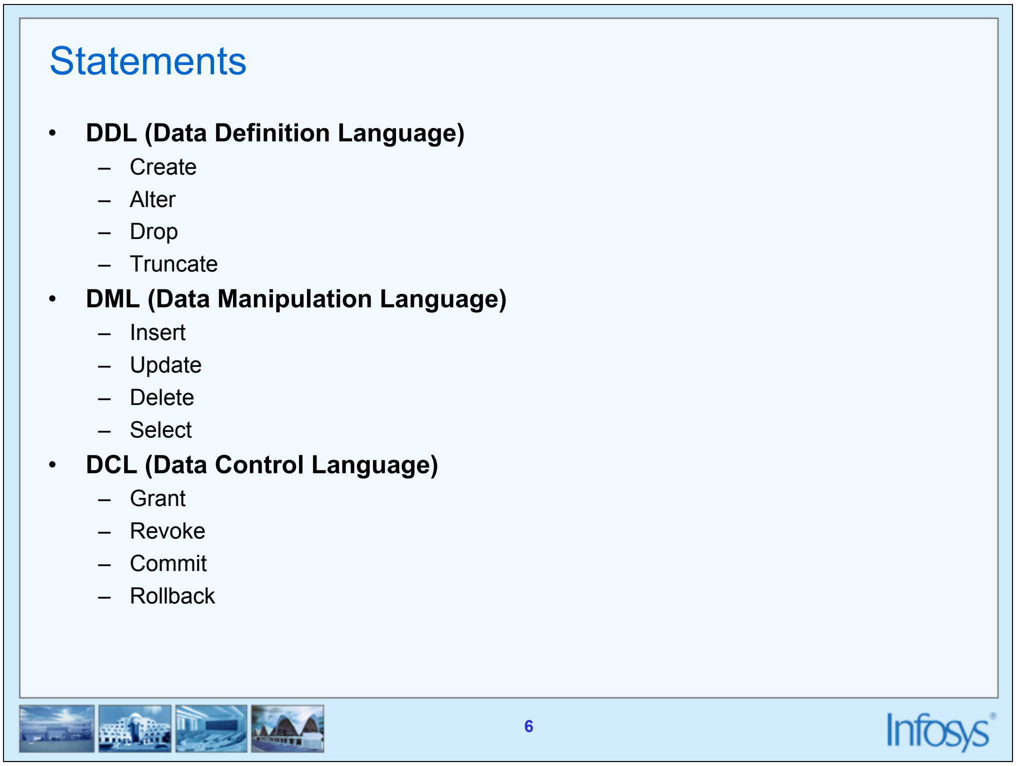
Structured Query Language (SQL)





Structured Query Language (SQL)

- **1979** Oracle Corporation introduces the first commercial RDBMS
- **1982** ANSI (American National Standards Institute) forms SQL Standards Committee
- **1983** IBM (International Business Machine) announces DB2 (a Database)
- **1986** ANSI (American National Standards Institute) SQL1 standard is approved
- **1987** ISO (International Organization for Standardization) SQL1 standard is approved
- **1992** ANSI (American National Standards Institute) SQL2 standard is approved
- **2000** Microsoft Corporation introduces SQL Server 2000, aimed at enterprise applications
- **2004** SQL: 2003 standard is published



Statements

- **DDL (Data Definition Language)**
  - Create
  - Alter
  - Drop
  - Truncate
- **DML (Data Manipulation Language)**
  - Insert
  - Update
  - Delete
  - Select
- **DCL (Data Control Language)**
  - Grant
  - Revoke
  - Commit
  - Rollback

SQL has three flavours of statements. The DDL, DML and DCL.

DDL is Data Definition Language statements. Some examples:

CREATE - to create objects in the database

ALTER - alters the structure of the database

DROP - delete objects from the database

TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed

COMMENT - add comments to the data dictionary

GRANT - gives user's access privileges to database

REVOKE - withdraw access privileges given with the GRANT command

DML is Data Manipulation Language statements. Some examples:

SELECT - retrieve data from the a database

INSERT - insert data into a table

UPDATE - updates existing data within a table

DELETE - deletes all records from a table, the space for the records remain

CALL - call a PL/SQL or Java subprogram

EXPLAIN PLAN - explain access path to data

LOCK TABLE - control concurrency

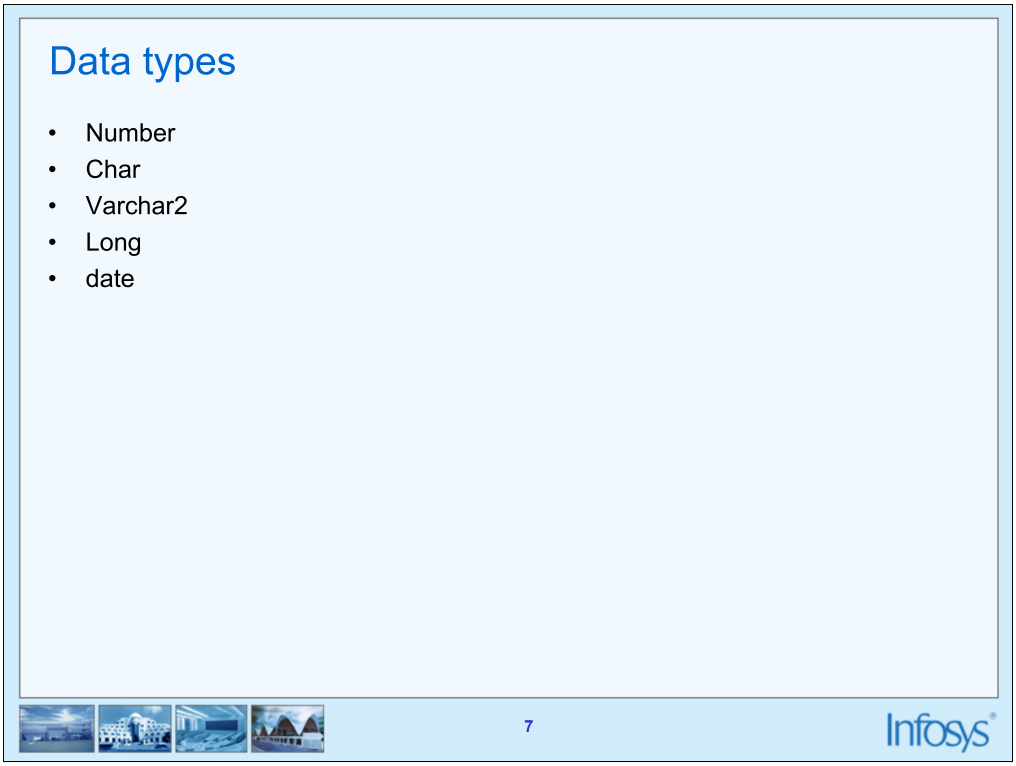
DCL is Data Control Language statements. Some examples:

COMMIT - save work done

SAVEPOINT - identify a point in a transaction to which you can later roll back

ROLLBACK - restore database to original since the last COMMIT

SET TRANSACTION - Change transaction options like what rollback segment to use



Data types

- Number
- Char
- Varchar2
- Long
- date

SQL supports various data types

Integers

Decimal numbers--- NUMBER, INTEGER .

Number is an oracle data type. Integer is an ANSI data type. Integer is equivalent of NUMBER(38)

The syntax for NUMBER is NUMBER(P,S) p is the precision and s is the scale. P can range from 1 to 38 and s from -84 to 127

Floating point numbers---- FLOAT

Fixed length character strings---- CHAR (len)

Fixed length character data of length len bytes. This should be used for fixed length data.

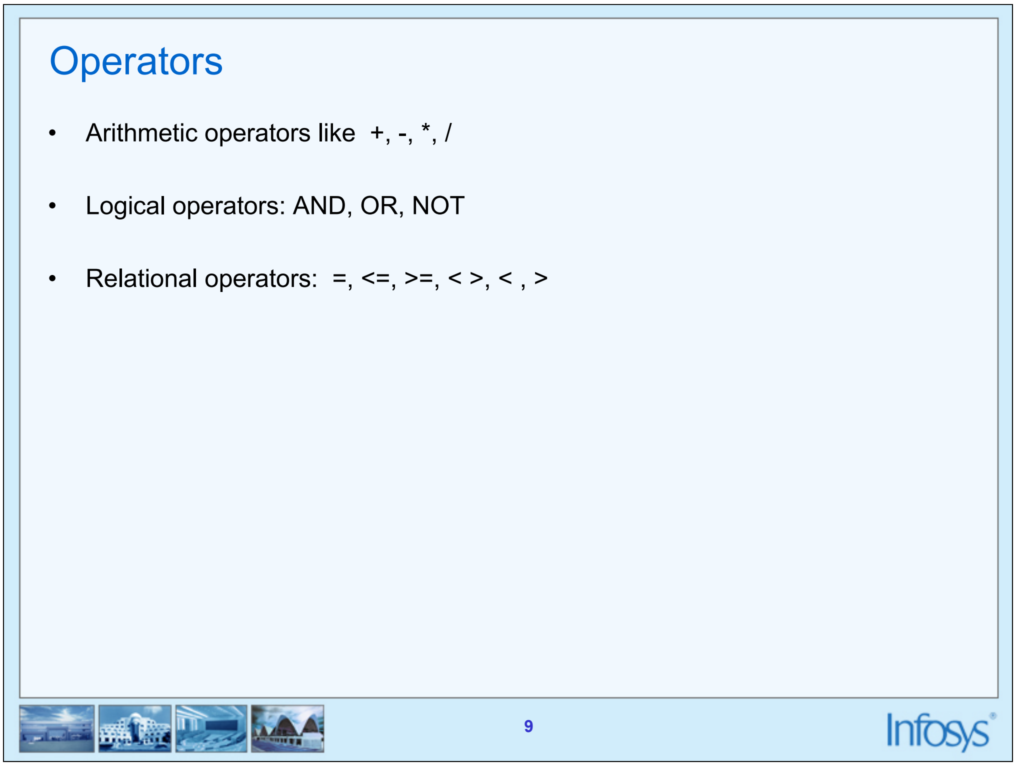
Variable length character strings --- Varchar2(len)

Variable length character string having maximum length *len* bytes. We must specify the size

Dates-----DATE

NULL

- Missing/unknown/inapplicable data represented as a **null** value
- NULL is not a data value. It is just an indicator that the value is unknown



Operators

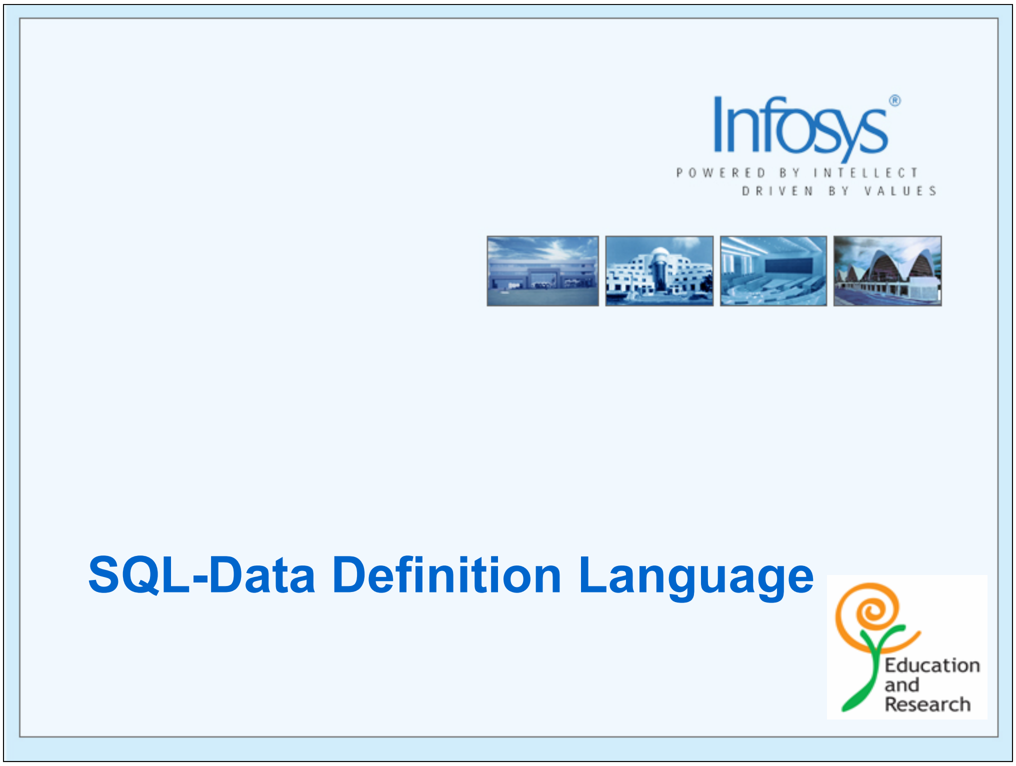
- Arithmetic operators like +, -, *, /
- Logical operators: AND, OR, NOT
- Relational operators: =, <=, >=, < >, < , >

The Arithmetic operators are used to calculate something like given in the example below:
Select * from employee where sal * 1.1 > 1000 ;

The logical operators are used to combine conditions like:

Select * from employee where (sal > 1000 AND age > 25);

The above two examples also illustrate use of relational operators



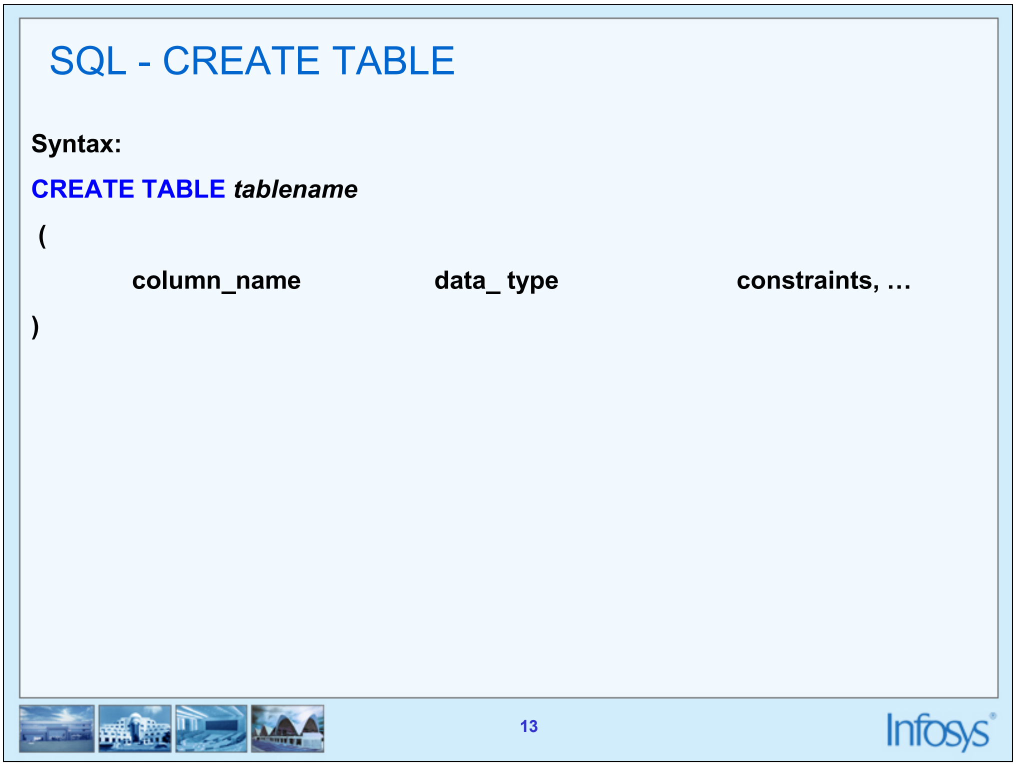
SQL-Data Definition Language

Types Of Constraints

- **Column Level**
- **Table**

Types Of Constraints

- Primary Key Constraint
- Foreign Key Constraint
- Unique Constraint
- Check Constraint



SQL - CREATE TABLE

Syntax:

```
CREATE TABLE tablename
 (
          column_name          data_ type          constraints, …
)
```

Used to create a table by defining its structure, the data type and name of the various columns, the relationships with columns of other tables etc.

Create Table (Contd…)

- **Implementing NOT NULL and Primary Key**

EXAMPLE :

```
CREATE TABLE Customer_Details(
    Cust_ID             Number(5)       CONSTRAINT Nnull1 NOT NULL,
    Cust_Last_Name      VarChar2(20)    CONSTRAINT Nnull2 NOT NULL,
    Cust_Mid_Name       VarChar2(4),
    Cust_First_Name     VarChar2(20),
    Account_No          Number(5)       CONSTRAINT Pkey1 PRIMARY KEY,
    Account_Type        VarChar2(10)    CONSTRAINT Nnull3 NOT NULL,
    Bank_Branch         VarChar2(25)    CONSTRAINT Nnull4 NOT NULL,
    Cust_Email          VarChar2(30)
);
```

Create Table (Contd…)

- **Implementing Composite Primary Key**

EXAMPLE :

```
CREATE TABLE Customer_Details(
    Cust_ID             Number(5)       CONSTRAINT Nnull7 NOT NULL,
    Cust_Last_Name      VarChar2(20)    CONSTRAINT Nnull8 NOT NULL,
    Cust_Mid_Name       VarChar2(4),
    Cust_First_Name     VarChar2(20),
    Account_No          Number(5)       CONSTRAINT Nnull9 NOT NULL,
    Account_Type        VarChar2(10)    CONSTRAINT Nnull10 NOT NULL,
    Bank_Branch         VarChar2(25)    CONSTRAINT Nnull11 NOT NULL,
    Cust_Email          VarChar2(30),
          CONSTRAINT PKey3 PRIMARY KEY(Cust_ID,Account_No)
);
```

Create Table (Contd…)

- **Implementation of Unique Constraint**

```
Create Table UnqTable(
ECode Number(6) Constraint PK11 Primary Key,
EName Varchar2(25) Constraint NNull18 NOT NULL,
EEmail Varchar2(25) Constraint Unq1 Unique
);
```

Create Table (Contd…)

- **Implementation of Primary Key and Foreign Key Constraints**

```
CREATE TABLE EMPLOYEE_MANAGER(
Employee_ID             Number(6) CONSTRAINT Pkey2 PRIMARY KEY,
Employee_Last_Name      VarChar2(25),
Employee_Mid_Name       VarChar2(5),
Employee_First_Name     VarChar2(25),
Employee_Email          VarChar2(35),
Department              VarChar2(10),
Grade                   Number(2),
MANAGER_ID              Number(6) CONSTRAINT Fkey2
                  REFERENCES EMPLOYEE_MANAGER(Employee_ID)
   );
```

Create Table (Contd…)

- **Implementation of Check Constraint**

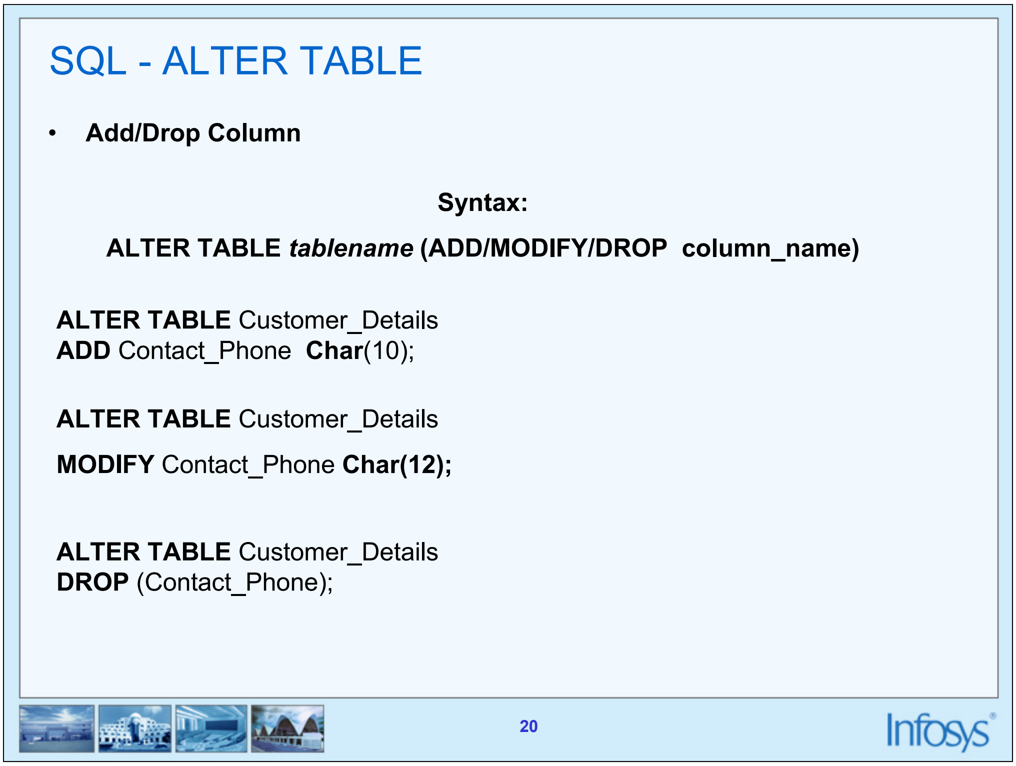
EXAMPLE :

```
CREATE TABLE  EMPLOYEE(
EmpNo NUMBER(5) CONSTRAINT PKey4 Primary Key,
EmpName Varchar(25) NOT NULL,
EmpSalary Number(7) Constraint chk Check (EmpSalary > 0 and EmpSalary <
    1000000)
)
```

Create Table (Contd…)

- **Implementation of Default**

```
CREATE TABLE  TABDEF(
    Ecode Number(4)  Not Null,
    Ename Varchar2(25) Not Null,
    ECity  char(10) DEFAULT ‘Mysore’
    )
```



SQL - ALTER TABLE

- **Add/Drop Column**

Syntax:

```
ALTER TABLE tablename (ADD/MODIFY/DROP  column_name)
```

```
ALTER TABLE Customer_Details
ADD Contact_Phone  Char(10);

ALTER TABLE Customer_Details

MODIFY Contact_Phone Char(12);


ALTER TABLE Customer_Details
DROP (Contact_Phone);
```

Used to modify the structure of a table by adding and removing columns

SQL - ALTER TABLE

- **Add/Drop Primary key**

```
ALTER TABLE Customer_Details
ADD CONSTRAINT Pkey1 PRIMARY KEY (Account_No);

ALTER TABLE Customer_Details
ADD CONSTRAINT Pkey2 PRIMARY KEY (Account_No, Cust_ID);

 ALTER TABLE Customer_Details
 DROP PRIMARY KEY;
 Or
 ALTER TABLE Customer_Details
 DROP CONSTRAINT Pkey1;
```

SQL - ALTER TABLE

- **Add/Drop Foreign key**

```
ALTER TABLE Customer_Transaction
ADD CONSTRAINT Fkey1 FOREIGN KEY (Account_No)
          REFERENCES Customer_Details (Account_No);


ALTER TABLE Customer_Transaction

DROP CONSTRAINT Fkey1
```

SQL - DROP TABLE

- **DROP TABLE**
  - Deletes table structure
  - Cannot be recovered
  - Use with caution

```
DROP TABLE UnqTable;
```

Truncate Table

- Deleting All Rows of a table

```
TRUNCATE TABLE Customer_Details
```

Index

- Indexing involves forming a two dimensional matrix completely independent of the table on which index is created.

- Here one column will hold the sorted data of the column which is been indexed

- Another column called the address field identifies the location of the record i.e. Row ID.

- Row Id indicates exactly where the record is stored in the table.

Index

- Syntax

```
CREATE  [UNIQUE] INDEX index-name on table-name  (column-name) [ ASC / DESC ]
```

- Index on a single column

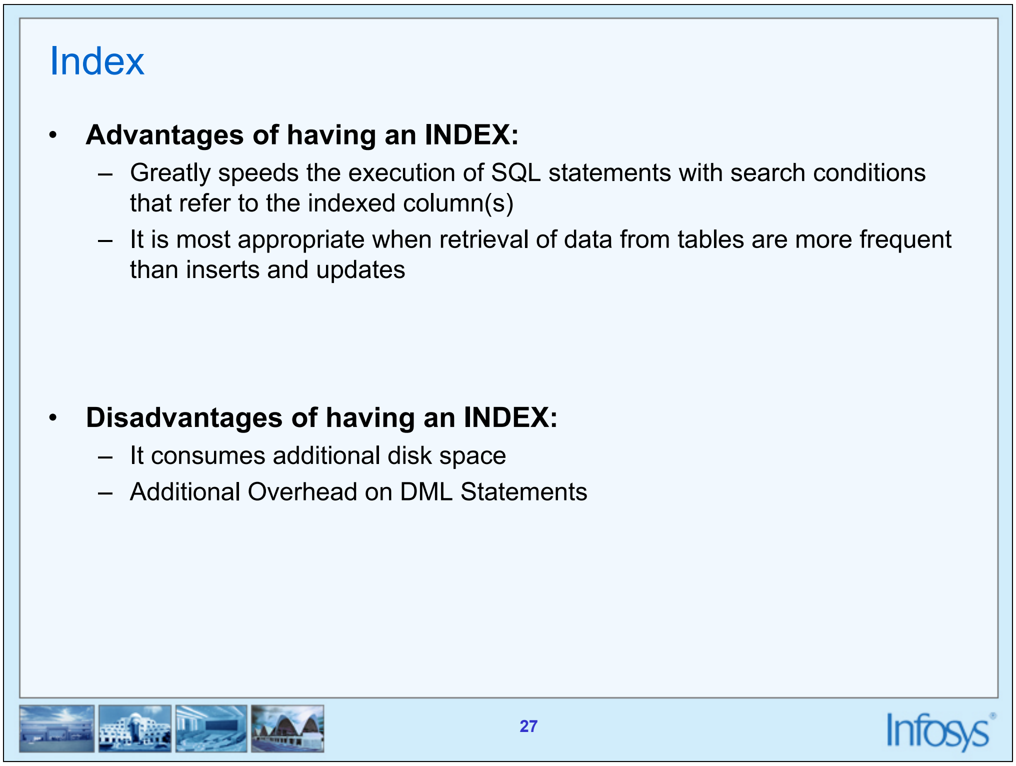
```
CREATE UNIQUE INDEX Cust_Idx
      ON Customer_Details (Cust_ID);
```

- Index on Multiple Column

```
CREATE UNIQUE INDEX ID_AccountNo_Idx
      ON Customer_Details (Cust_ID, Account_No);
```

- Drop a Index

```
DROP INDEX ID_AccountNo_Idx;
```



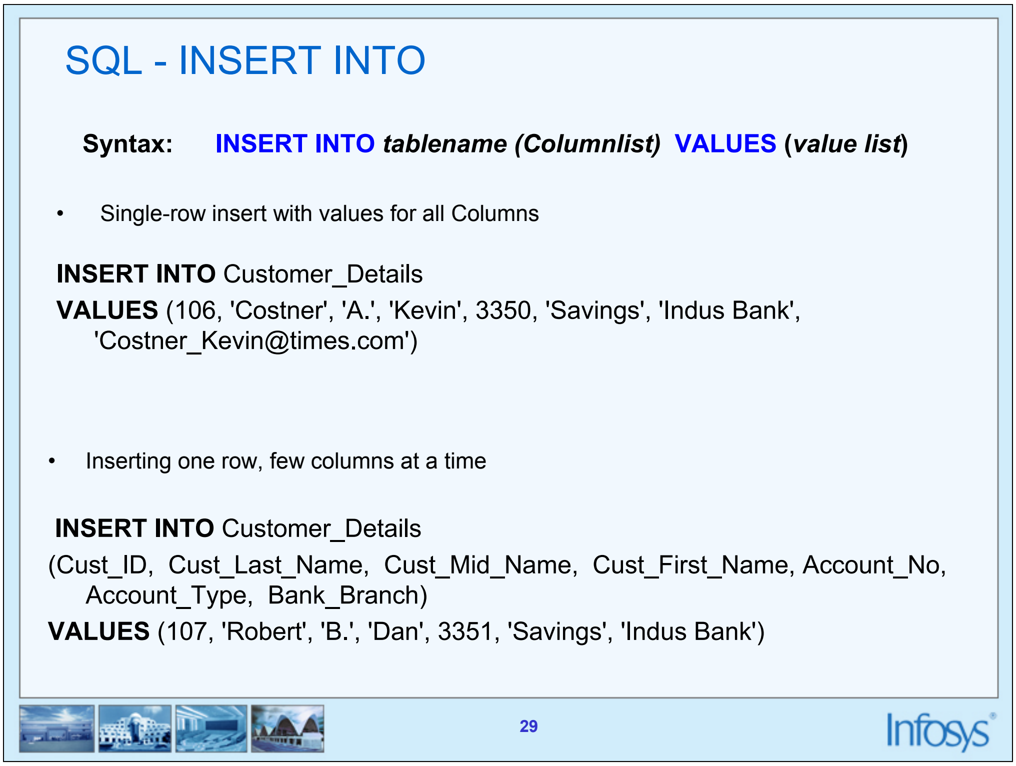
Index

- **Advantages of having an INDEX:**
  - Greatly speeds the execution of SQL statements with search conditions that refer to the indexed column(s)
  - It is most appropriate when retrieval of data from tables are more frequent than inserts and updates

- **Disadvantages of having an INDEX:**
  - It consumes additional disk space
  - Additional Overhead on DML Statements

SQL-DML

Here we will discuss about commands using which data from the tables would be extracted and updated in different ways



SQL - INSERT INTO

Syntax: **INSERT INTO** ***tablename (Columnlist)*** **VALUES (*value list*)**

- Single-row insert with values for all Columns

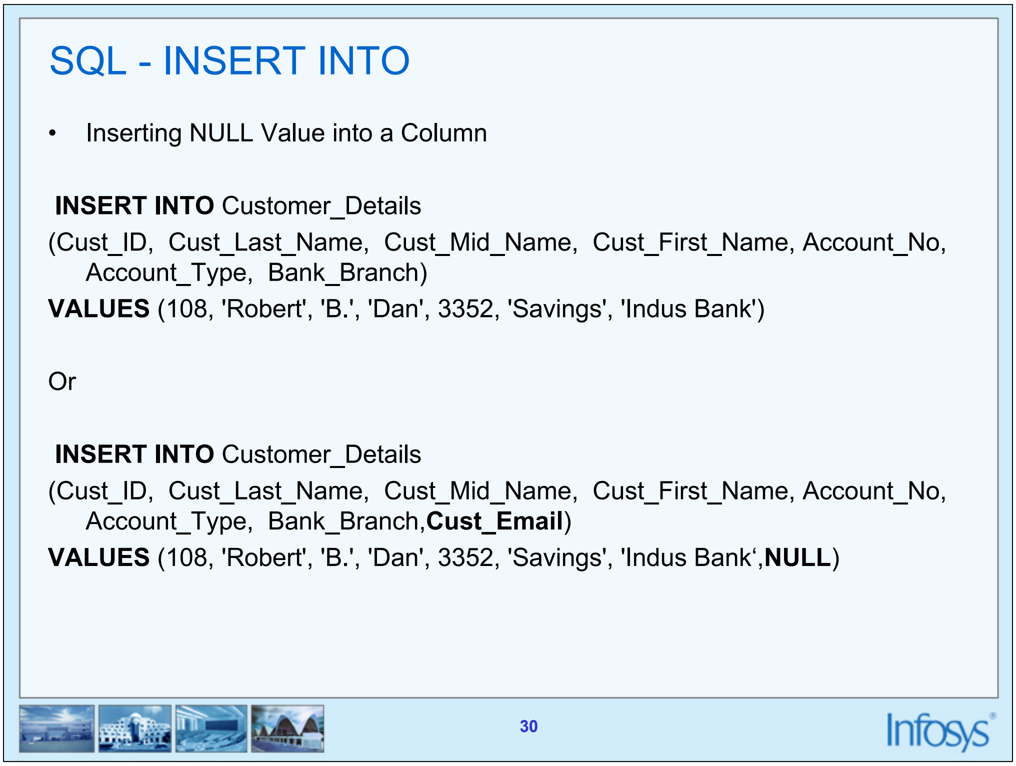
INSERT INTO Customer_Details
VALUES (106, 'Costner', 'A.', 'Kevin', 3350, 'Savings', 'Indus Bank', 'Costner_Kevin@times.com')

- Inserting one row, few columns at a time

INSERT INTO Customer_Details
(Cust_ID, Cust_Last_Name, Cust_Mid_Name, Cust_First_Name, Account_No, Account_Type, Bank_Branch)
VALUES (107, 'Robert', 'B.', 'Dan', 3351, 'Savings', 'Indus Bank')

In the first format, we would pass values for all the columns in exactly the same order in which they appear in the table

When we wish to insert values only for few selected columns. For e.g in a Customer table, we may know only the Cust_Id, Cust_Last_Name, Cust_Mid_Name, Cust_First_Name, Account_No, Account_Type and Bank_Branch but not the emailid. So, we may insert only values for Cust_Id, Cust_Last_Name, Cust_Mid_Name, Cust_First_Name, Account_No, Account_Type and Bank_Branch columns in this case. The value of the remaining column will be represented as NULL by default.



SQL - INSERT INTO

- Inserting NULL Value into a Column

```
 INSERT INTO Customer_Details
(Cust_ID,  Cust_Last_Name,  Cust_Mid_Name,  Cust_First_Name, Account_No,
    Account_Type,  Bank_Branch)
VALUES (108, 'Robert', 'B.', 'Dan', 3352, 'Savings', 'Indus Bank')
```

Or

```
 INSERT INTO Customer_Details
(Cust_ID,  Cust_Last_Name,  Cust_Mid_Name,  Cust_First_Name, Account_No,
    Account_Type,  Bank_Branch,Cust_Email)
VALUES (108, 'Robert', 'B.', 'Dan', 3352, 'Savings', 'Indus Bank‘,NULL)
```

SQL - INSERT INTO

- Inserting Many rows from a Different Table

```
INSERT INTO OldCust_details
(Account_No, Transaction_Date,Total_Available_Balance_in_Dollars)
SELECT Account_No,Transaction_Date,Total_Available_Balance_in_Dollars
            From Customer_Transaction
                        WHERE Total_Available_Balance_in_Dollars > 10000.00
```

SQL - DELETE FROM

- With or without WHERE clause

Syntax: DELETE FROM *tablename* WHERE *condition*

Deleting All Rows

DELETE FROM Customer_Details

Deleting Specific Rows

DELETE

FROM Customer_Details

WHERE Cust_ID = 102

Difference Between Delete and Truncate

DELETE	TRUNCATE
Data can be recovered	Data cannot be recovered.
DML statement	DDL statement
DELETE does not do so	TRUNCATE releases the memory occupied by the records of the table

SQL - UPDATE

Syntax:

UPDATE *tablename* SET *column_name* =*value* [WHERE *condition*]

Updating All Rows

```
UPDATE Customer_Fixed_Deposit
           SET Rate_of_Interest_in_Percent = NULL;
```

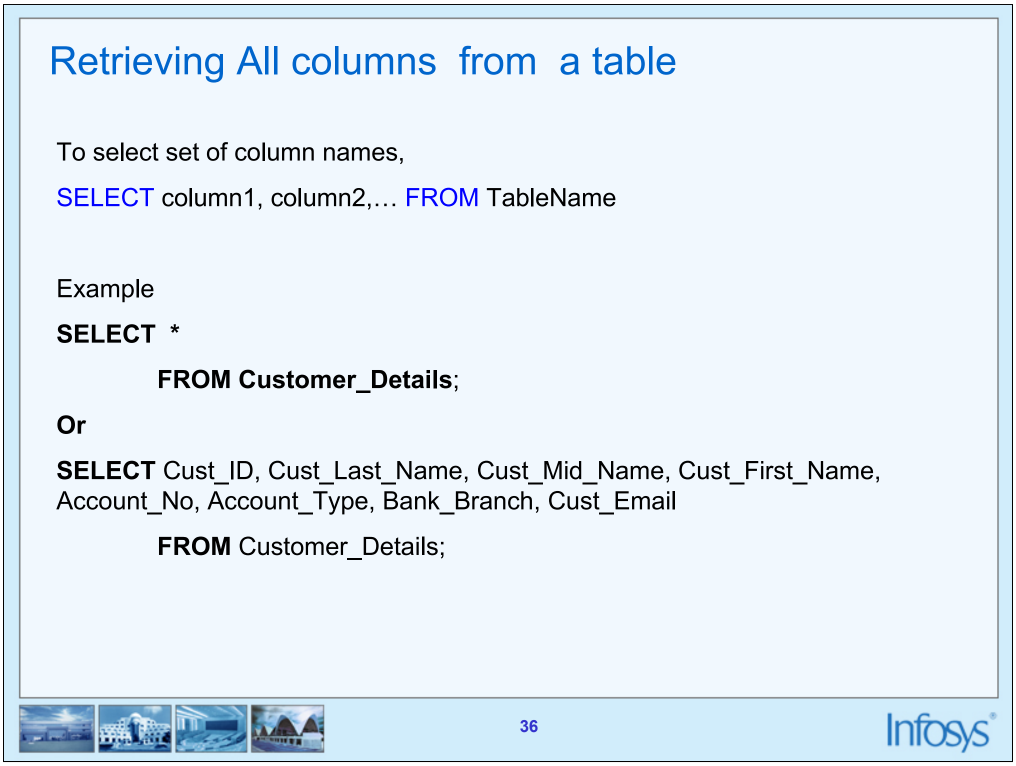
Updating Particular rows

```
UPDATE Customer_Fixed_Deposit
           SET Rate_of_Interest_in_Percent = 7.3
                      WHERE Amount_in_Dollars > 3000;
```

SQL - UPDATE

- Updating Multiple Columns

```
UPDATE Customer_Fixed_Deposit
    SET Cust_Email = ‘Quails_Jack@rediffmail.com’ ,
     Rate_of_Interest_in_Percent = 7.3
            WHERE Cust_ID = 104
```



Retrieving All columns from a table

To select set of column names,

```
SELECT column1, column2,… FROM TableName
```

Example

```
SELECT  *
          FROM Customer_Details;
```

Or

```
SELECT Cust_ID, Cust_Last_Name, Cust_Mid_Name, Cust_First_Name,
Account_No, Account_Type, Bank_Branch, Cust_Email
          FROM Customer_Details;
```

Examples:

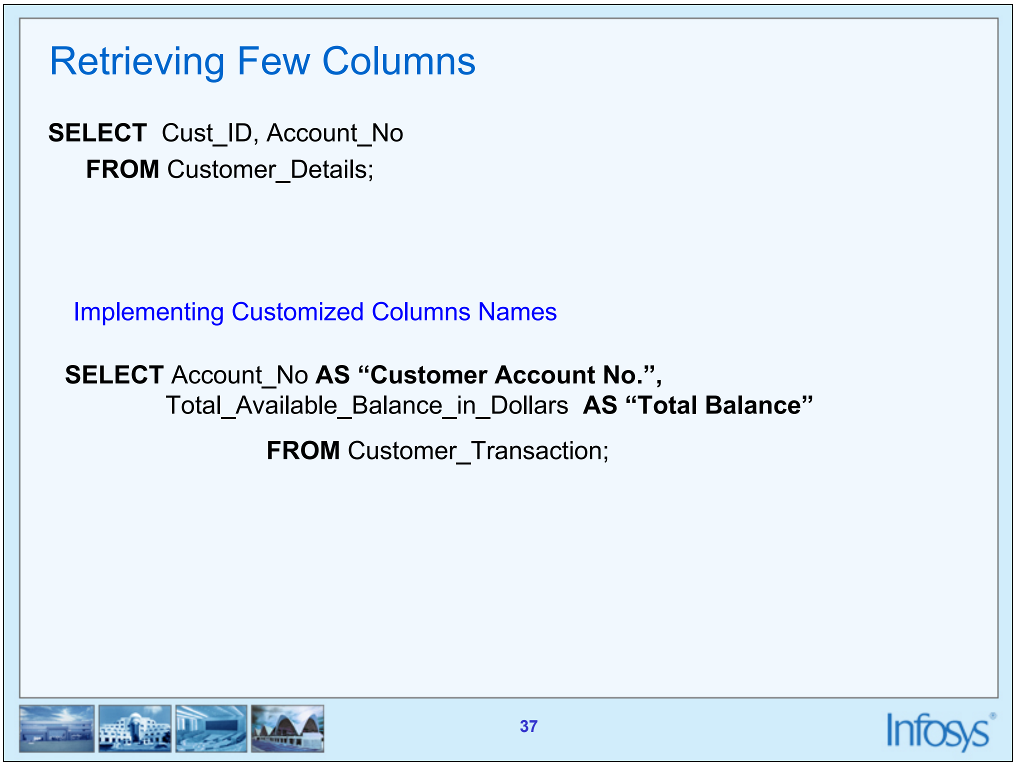
SELECT Cust_First_Name FROM **Customer_Details**;

Get the   and bank branch of all the

SELECT Cust_First_Name, Bank_Branch FROM S

Customer_Details

SELECT   * FROM **Customer_Details;**

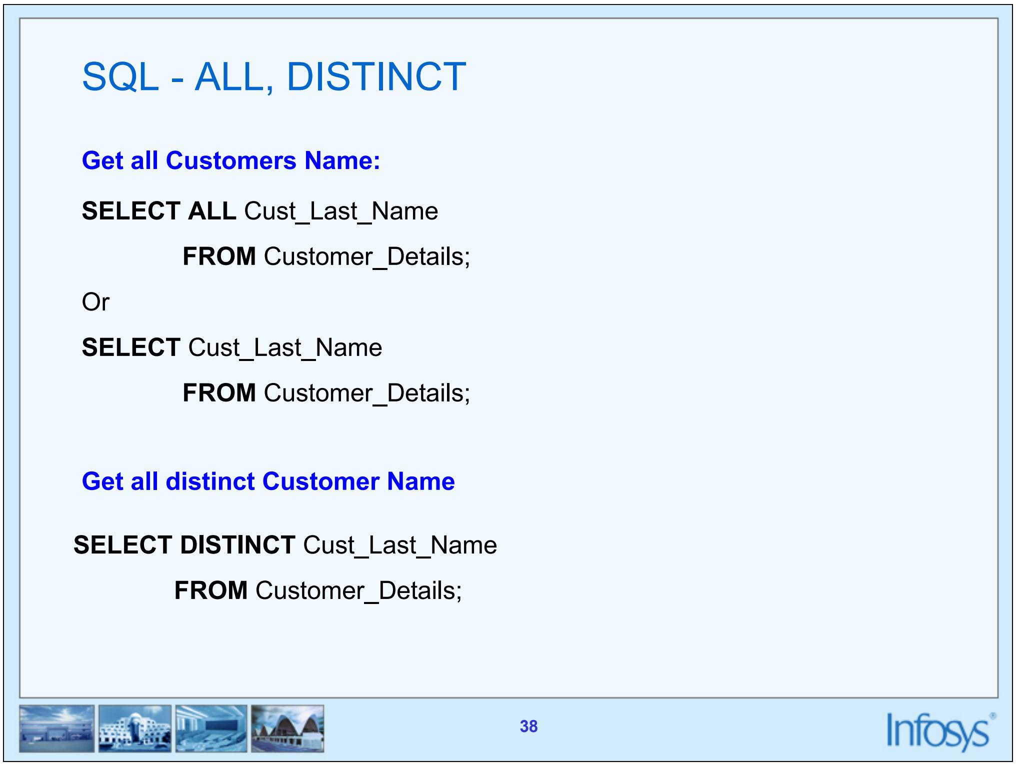


Retrieving Few Columns

```
SELECT  Cust_ID, Account_No
    FROM Customer_Details;
```

Implementing Customized Columns Names

```
SELECT Account_No AS "Customer Account No.",
           Total_Available_Balance_in_Dollars  AS "Total Balance"
                      FROM Customer_Transaction;
```



SQL - ALL, DISTINCT

Get all Customers Name:

```
SELECT ALL Cust_Last_Name
            FROM Customer_Details;
Or
SELECT Cust_Last_Name
            FROM Customer_Details;
```

Get all distinct Customer Name

```
SELECT DISTINCT Cust_Last_Name
            FROM Customer_Details;
```

Distinct will filter repetitive occurrence of a particular value

Retrieving a subset of rows

- **For retrieval of rows based on some condition, the syntax is**

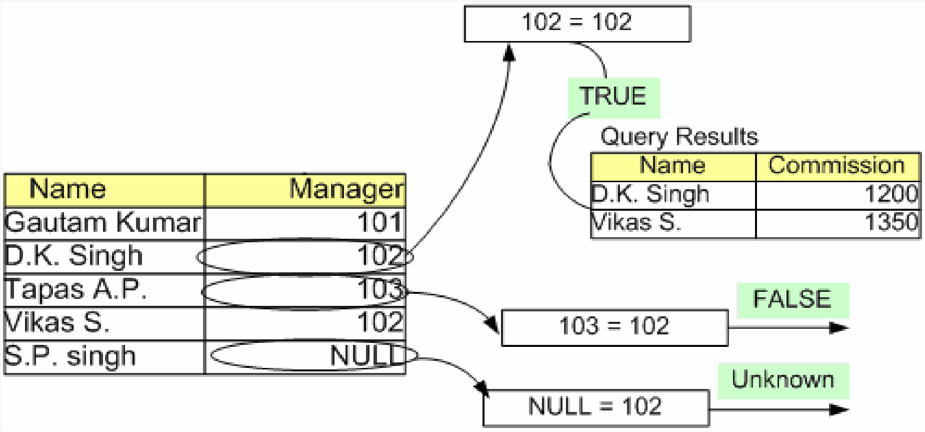
```
SELECT  COL1,COL2,.........

          FROM   TABLE NAME

                    WHERE  < SEARCH CONDITION>
```

Retrieving a subset of rows (Working of WHERE Clause)

Problem Statement: To select rows which have 102 in the Manager column.



Name	Manager
Gautam Kumar	101
D.K. Singh	102
Tapas A.P.	103
Vikas S.	102
S.P. singh	NULL

Query Results

Name	Commission
D.K. Singh	1200
Vikas S.	1350

Row selection with the WHERE clause

Relational operators

- ***List all customers with an account balance > $10000***

```
SELECT Account_No, Total_Available_Balance_in_Dollars
        FROM Customer_Transaction
                WHERE Total_Available_Balance_in_Dollars > 10000.00;
```

- ***List the Cust_ID, Account_No of 'Graham'***

```
SELECT Cust_ID, Account_No
        FROM Customer_Details
                WHERE Cust_First_Name = 'Graham';
```

Relational operators = , < , > , <= , >= , != or < >

Infosys®

Relational operators

- *List all Account_No where Total_Available_Balance_in_Dollars is atleast $10000.00*

```
SELECT Account_No
FROM Customer_Transaction
WHERE Total_Available_Balance_in_Dollars >= 10000.00;
```

Logical operators

- *List all Cust_ID, Cust_Last_Name where Account_type is ‘Savings’ and Bank_Branch is ‘Capital Bank’.*

```
SELECT Cust_ID, Cust_Last_Name

FROM Customer_Details

WHERE Account_Type = ‘Savings’ AND Bank_Branch = ‘Capital Bank’;
```

- *List all Cust_ID, Cust_Last_Name where neither Account_type is ‘Savings’ and nor Bank_Branch is ‘Capital Bank’*

```
SELECT Cust_ID, Cust_Last_Name

FROM Customer_Details

WHERE NOT Account_Type = ‘Savings’ AND

          NOT Bank_Branch = ‘Capital Bank’;
```

Logical operators

- ***List all Cust_ID, Cust_Last_Name where either Account_type is ‘Savings’ or Bank_Branch is ‘Capital Bank’.***

```
SELECT Cust_ID, Cust_Last_Name
FROM Customer_Details
WHERE Account_Type = ‘Savings’  OR  Bank_Branch = ‘Capital Bank’;
```

Logical operator: AND, OR, and NOT

Retrieval using BETWEEN

test-expression **[NOT] BETWEEN** low-expression **AND** high-expression

List all Account_Nos with balance in the range $10000.00 to $20000.00.

```
SELECT Account_No
FROM Customer_Transaction
WHERE Total_Available_Balance_in_Dollars >= 10000.00
            AND Total_Available_Balance_in_Dollars <= 20000.00;
Or
SELECT Account_No
FROM Customer_Transaction
WHERE Total_Available_Balance_in_Dollars
              BETWEEN 10000.00 AND 20000.00;
```

Retrieval using IN

test-expression **[NOT] IN** (constant1, constant2…………)

List all customers who have account in Capital Bank or Indus Bank.

```
SELECT Cust_ID
FROM Customer_Details
WHERE Bank_Branch = ‘Capital Bank’
            OR Bank_Branch =  ‘Indus Bank’;
Or
SELECT Cust_ID
FROM Customer_Details
WHERE Bank_Branch IN (‘Capital Bank’, ‘Indus Bank’);
```

Retrieval using LIKE

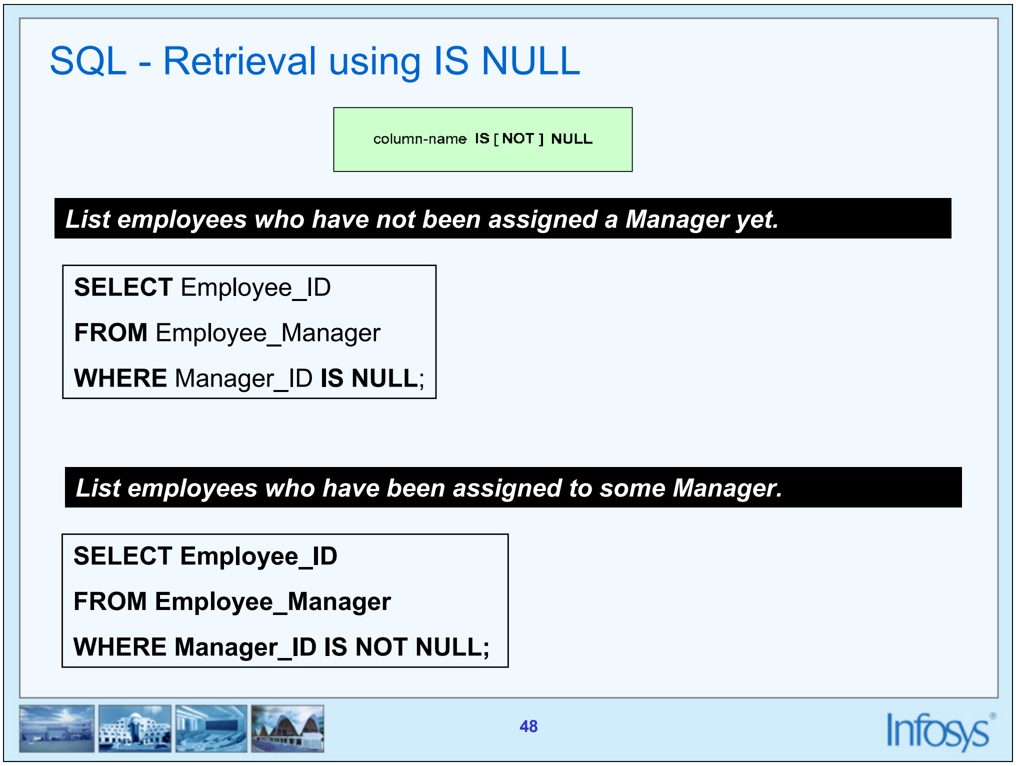
Column-name **[NOT] LIKE** pattern **ESCAPE** escape-character

List all Accounts where the Bank_Branch begins with a ‘C’ and has ‘a’ as the second character

```
SELECT Cust_ID, Cust_Last_Name, Account_No
FROM Customer_Details
WHERE Bank_Branch LIKE ‘Ca%’;
```

List all Accounts where the Bank_Branch column has ‘a’ as the second character.

```
SELECT Cust_ID, Cust_Last_Name, Account_No
FROM Customer_Details
WHERE Bank_Branch LIKE ‘_a%’;
```



SQL - Retrieval using IS NULL

```
column-name IS [ NOT ] NULL
```

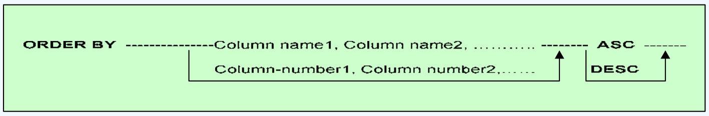
List employees who have not been assigned a Manager yet.

```
SELECT Employee_ID
FROM Employee_Manager
WHERE Manager_ID IS NULL;
```

List employees who have been assigned to some Manager.

```
SELECT Employee_ID
FROM Employee_Manager
WHERE Manager_ID IS NOT NULL;
```

SQL - Sorting your results (ORDER BY)


ORDER BY ---------------Column name1, Column name2, ... -------- ASC -------

Column-number1, Column number2,......

DESC

List the customers account numbers and their account balances, in the increasing order of the balance

```
SELECT Account_No, Total_Available_Balance_in_Dollars
          FROM Customer_Transaction
                    ORDER BY Total_Available_Balance_in_Dollars;
```

- *by default the order is ASCENDING*

Retrieval using ORDER BY

List the customers and their account numbers in the decreasing order of the account numbers.

```
SELECT  Cust_Last_Name, Cust_First_Name, Account_No
          FROM Customer_Details
                    ORDER BY 3 DESC;
```

Retrieval using ORDER BY

List the customers and their account numbers in the decreasing order of the Customer Last Name and increasing order of account numbers.

```
SELECT  Cust_Last_Name, Cust_First_Name, Account_No
            FROM Customer_Details
            ORDER BY Cust_Last_Name DESC, Account_No;
Or
SELECT  Cust_Last_Name, Cust_First_Name, Account_No
            FROM Customer_Details
            ORDER BY 1 DESC, 3;
```

Infosys®

POWERED BY INTELLECT
DRIVEN BY VALUES

Aggregate Functions

Education
and
Research

SQL - Aggregate functions

- Used when information you want to extract from a table has to do with the data in the entire table taken as a set.
- Aggregate functions are used in place of column names in the SELECT statement
- The aggregate functions in sql are :

SUM( ) , AVG( ) , MAX( ) , MIN( ), COUNT( )

```
SUM ( [ DISTINCT ] column-name / expression )
AVG ( [ DISTINCT ] column-name / expression )
MIN ( expression)
MAX ( expression )
COUNT ( [ DISTINCT ] column-name )
COUNT ( *)
```

Aggregate function - MIN

- Returns the smallest value that occurs in the specified column
- Column need not be numeric type

List the minimum account balance.

```
SELECT MIN (Total_Available_Balance_in_Dollars)
          FROM Customer_Transaction;
```

Aggregate function - MAX

- Returns the largest value that occurs in the specified column
- Column need not be numeric type

- Example:

List the maximum account balance.

```
SELECT MAX (Total_Available_Balance_in_Dollars)
FROM Customer_Transaction;
```

Aggregate function - AVG

- Returns the average of all the values in the specified column
- Column must be numeric data type

Example:

List the average account balance of customers.

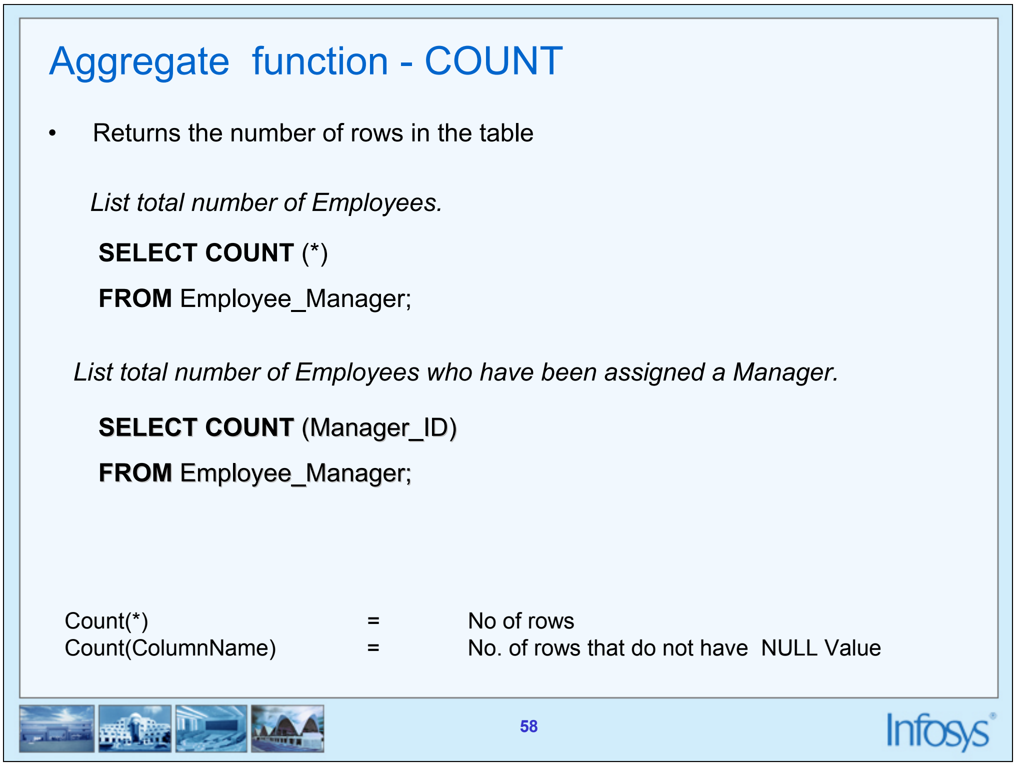
```
SELECT AVG (Total_Available_Balance_in_Dollars)
FROM Customer_Transaction;
```

Aggregate function - SUM

- Adds up the values in the specified column
- Column must be numeric data type
- Value of the sum must be within the range of that data type
- **Example:**

List the minimum and Sum of all account balance.

```
SELECT MIN (Total_Available_Balance_in_Dollars),
          SUM (Total_Available_Balance_in_Dollars)
FROM Customer_Transaction;
```



Aggregate function - COUNT

- Returns the number of rows in the table

List total number of Employees.

SELECT COUNT (*)

FROM Employee_Manager;

List total number of Employees who have been assigned a Manager.

SELECT COUNT (Manager_ID)

FROM Employee_Manager;

Count(*)	=	No of rows
Count(ColumnName)	=	No. of rows that do not have NULL Value

Aggregate function - COUNT

List total number of account holders in the ‘Capital Bank’ Branch.

```
SELECT COUNT (*)
FROM Customer_Details
WHERE Bank_Branch = ‘Capital Bank’;
```

List total number of unique Customer Last Names.

```
SELECT COUNT (DISTINCT Cust_Last_Name)
FROM Customer_Details;
```

Count(*)	=	No of rows
Count(ColumnName)	=	No. of rows that do not have NULL Value

Summary of basic DDL and DML

- Create , Alter and Drop are the DDL commands

- Update, Insert into, Delete from are the basic DML commands that add or remove data from tables

- Select statement in its various flavours is used to retrieve information from the table

- Aggregate functions work on all the rows of the table taken as a group (based on some condition optionally)

Thank You!