1	Hemanth	34	13000

What is Field ?

A table consists of several records(row), each record can be broken into several smaller entities known as**Fields**. The above **Employee** table consist of four fields, **ID**, **Name**, **Age** and **Salary**.

What is a Column/Attribute ?

In **Relational** table, a column is a set of value of a particular type. The term **Attribute** is also used to represent a column.

- A column header is called an attribute

For example, in Employee table, Name is a column that represents names of employee.

Name
Adam
Alex
Stuart
Ross

importance of NULL values

- An important concept is that if NULL values, which are used to represent the values of attributes that may be unknown or may not apply to a tuple

Relational Model Notation

- An attribute A can be qualified with the relation name R to which it belongs by using the dot notation R.A
- For example, STUDENT.Name or STUDENT.Age

Database Keys

Keys are very important part of Relational database. They are used to establish and identify relation between tables. They also ensure that each record within a table can be uniquely identified by combination of one or more fields within a table.

Super Key

Super Key is defined as a set of one or more attributes within a table that uniquely identifies each record within a table(identify all other attributes uniquely). Super Key is a superset of Candidate key.

For Example, We are having table

```
Book (BookId, BookName, Author)
```

So in this table we can have following super keys

```
¬   (BookId)
¬   (BookId,BookName)
¬   (BookId, BookName, Author)
¬   (BookId, Author)
¬   (BookName, Author)
```

Each super key is able to uniquely identify each tuple (record).

Candidate Key

Candidate keys are defined as the set of fields from which primary key can be selected.

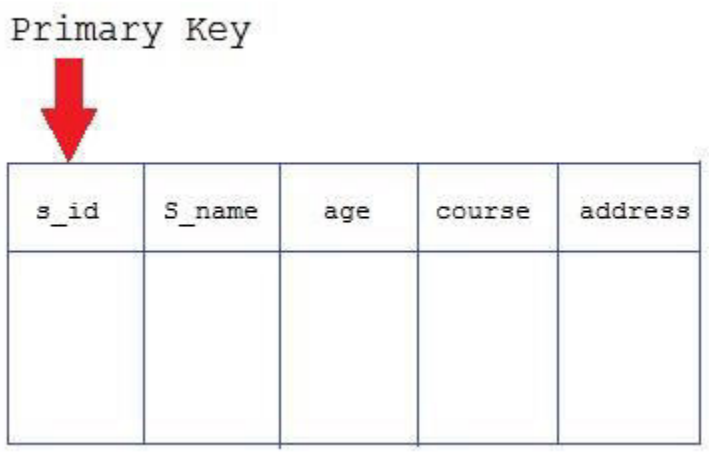
Candidate keys are a super key which are not having any redundant attributes. In other words candidate keys are minimal super keys. For Example, In above illustration

```
¬   (BookId)
¬   (BookName,Author)
```

These two keys can be candidate keys, as remaining keys are having redundant attributes.

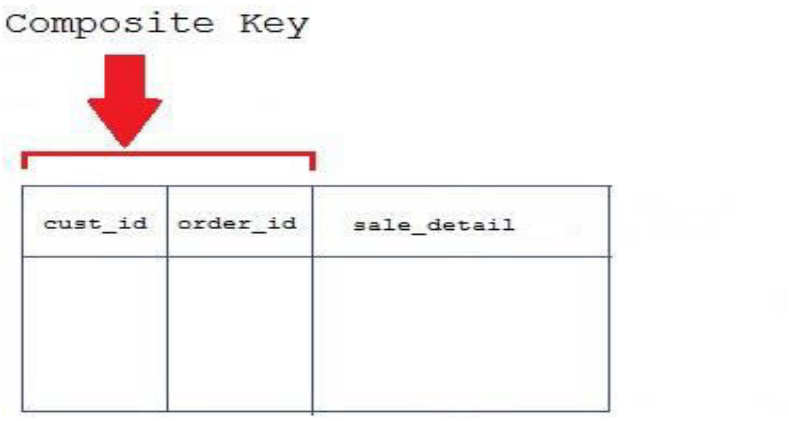
Primary Key

Primary key is a candidate key that is most appropriate to become main key of the table. It is a key that uniquely identify each record in a table.



Composite Key

Key that consist of two or more attributes that uniquely identify an entity occurance is called **Composite key**. But any attribute that makes up the **Composite key** is not a simple key in its own



Secondary or Alternative key

The candidate key which are not selected for primary key are known as secondary keys or alternative keys

---

Non-key Attribute

Non-key attributes are attributes other than **candidate key** attributes in a table.

Non-prime Attribute

Non-prime Attributes are attributes other than **Primary attribute**.

Constraints and their importance:

Constraints are rules used to limit the type of data that can go into a table, to maintain the accuracy and integrity of the data inside table.

Constraints can be divided into following two types,

- **Column level constraints :** limits only column data
- **Table level constraints :** limits whole table data

Constraints are used to make sure that the integrity of data is maintained in the database. Following are the most used constraints that can be applied to a table.

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT

NOT NULL Constraint

NOT NULL constraint restricts a column from having a NULL value. Once **NOT NULL** constraint is applied to a column, you cannot pass a null value to that column. It enforces a column to contain a proper value. One important point to note about NOT NULL constraint is that it cannot be defined at table level.

Example using NOT NULL constraint

```
CREATE table Student(s_id int NOT NULL, Name varchar(60), Age int);
```

The above query will declare that the **s_id** field of **Student** table will not take NULL value.

UNIQUE Constraint

UNIQUE constraint ensures that a field or column will only have unique values. A UNIQUE constraint field will not have duplicate data. UNIQUE constraint can be applied at column level or table level.

Example using UNIQUE constraint when creating a Table (Table Level)

```
CREATE table Student(s_id int NOT NULL UNIQUE, Name varchar(60), Age int);
```

The above query will declare that the **s_id** field of **Student** table will only have unique values and wont take NULL value.

Example using UNIQUE constraint after Table is created (Column Level)

```
ALTER table Student add UNIQUE(s_id);
```

The above query specifies that **s_id** field of **Student** table will only have unique value.

Primary Key Constraint

Primary key constraint uniquely identifies each record in a database. A Primary Key must contain unique value and it must not contain null value. Usually Primary Key is used to index the data inside the table.

Example using PRIMARY KEY constraint at Table Level

```
CREATE table Student (s_id int PRIMARY KEY, Name varchar(60) NOT NULL, Age int);
```

The above command will creates a PRIMARY KEY on the `s_id`.

Example using PRIMARY KEY constraint at Column Level

```
ALTER table Student add PRIMARY KEY (s_id);
```

The above command will creates a PRIMARY KEY on the `s_id`.

Foreign Key Constraint

FOREIGN KEY is used to relate two tables. FOREIGN KEY constraint is also used to restrict actions that would destroy links between tables. To understand FOREIGN KEY, let's see it using two table.

Customer_Detail Table :

c_id	Customer_Name	address
101	Adam	Noida
102	Alex	Delhi
103	Stuart	Rohtak

Order_Detail Table :

Order_id	Order_Name	c_id
10	Order1	101
11	Order2	103
12	Order3	102

In **Customer_Detail** table, c_id is the primary key which is set as foreign key in **Order_Detail** table. The value that is entered in c_id which is set as foreign key in **Order_Detail** table must be present in **Customer_Detail**table where it is set as primary key. This prevents invalid data to be inserted into c_id column of **Order_Detail**table.

Example using FOREIGN KEY constraint at Table Level

```
CREATE table Order_Detail(order_id int PRIMARY KEY,
order_name varchar(60) NOT NULL,
c_id int FOREIGN KEY REFERENCES Customer_Detail(c_id));
```

In this query, c_id in table Order_Detail is made as foriegn key, which is a reference of c_id column of Customer_Detail.

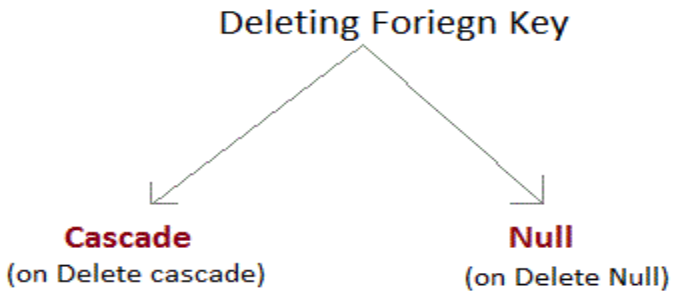
Example using FOREIGN KEY constraint at Column Level

```
ALTER table Order_Detail add FOREIGN KEY (c_id) REFERENCES Customer_Detail(c_id);
```

Behaviour of Foriegn Key Column on Delete

There are two ways to maintin the integrity of data in Child table, when a particular record is deleted in main table. When two tables are connected with Foriegn key, and certain data in the main table is

deleted, for which record exit in child table too, then we must have some mechanism to save the integrity of data in child table.



- **On Delete Cascade :** This will remove the record from child table, if that value of foriegn key is deleted from the main table.
- **On Delete Null :** This will set all the values in that record of child table as NULL, for which the value of foriegn key is eleted from the main table.
- If we don't use any of the above, then we cannot delete data from the main table for which data in child table exists. We will get an error if we try to do so.

```
ERROR : Record in child table exist
```

CHECK Constraint

CHECK constraint is used to restrict the value of a column between a range. It performs check on the values, before storing them into the database. Its like condition checking before saving data into a column.

Example using CHECK constraint at Table Level

```
create table Student(s_id int NOT NULL CHECK(s_id > 0),
Name varchar(60) NOT NULL,
Age int);
```

The above query will restrict the s_id value to be greater than zero.

Example using CHECK constraint at Column Level

```
ALTER table Student add CHECK(s_id > 0);
```

Default Constraint

- **Default**: sets a default value for the column. If you specify a column called date_added with DEFAULT GETDATE() then every row you insert will automatically have the date/time it was created as part of the row.

SQL: structured query language

SQL is a standard language for accessing and manipulating databases.

What is SQL?

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL is an ANSI (American National Standards Institute) standard

What Can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

SQL is a Standard - BUT....

Although SQL is an ANSI (American National Standards Institute) standard, there are different versions of the SQL language.

However, to be compliant with the ANSI standard, they all support at least the major commands (such as SELECT, UPDATE, DELETE, INSERT, WHERE) in a similar manner.

RDBMS is the basis for SQL, and for all modern database systems such as MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

The data in RDBMS is stored in database objects called tables.

Database Tables

A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"). Tables contain records (rows) with data.

In this tutorial we will use the well-known Northwind sample database (included in MS Access and MS SQL Server).

Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

The table above contains five records (one for each customer) and seven columns (CustomerID, CustomerName, ContactName, Address, City, PostalCode, and Country).

Keep in Mind That...

- SQL is NOT case sensitive: select is the same as SELECT

SQL Statements

Most of the actions you need to perform on a database are done with SQL statements.

The following SQL statement selects all the records in the "Customers" table:

Example

```
SELECT * FROM Customers;
```

Semicolon after SQL Statements?

Some database systems require a semicolon at the end of each SQL statement.

Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.

Some of The Most Important SQL Commands

- **SELECT** - extracts data from a database
- **UPDATE** - updates data in a database
- **DELETE** - deletes data from a database
- **INSERT INTO** - inserts new data into a database
- **CREATE DATABASE** - creates a new database
- **ALTER DATABASE** - modifies a database
- **CREATE TABLE** - creates a new table
- **ALTER TABLE** - modifies a table
- **DROP TABLE** - deletes a table
- **CREATE INDEX** - creates an index (search key)
- **DROP INDEX** - deletes an index

simple database schema:

An sql schema is identified by a schema name, and includes an authorization identifier to indicate the user or account who owns the schema, as well as descriptors for each element in the schema. Schema elements include tables,constraints,views,domains, and other constructs(such as authorization grants) that describe the schema

A schema is created via the CREATE SCHEMA statement which can include all schema elements definitions

Example:

CREATE SCHEMA COMPANY **AUTHORIZATION** hemanth**;**

NOTE: **Data Definition** Language (DDL) statements -used to define the database structure or schema.(refer DDL commands)

Data types

- SQL data type is an attribute that specifies type of data of any object. Each column, variable and expression has related data type in SQL.
- You would use these data types while creating your tables. You would choose a particular data type for a table column based on your requirement.
- The basic data types include

1. Numeric data types include integer numbers of various sizes(INTEGER or INT,SMALLINT) And floating-point(real) numbers of various precision(FLOAT or REAL, and DOUBLE PRECISION) Formatted numbers can be declared using DECIMAL(i,j)-or DEC(i,j) or NUMERIC(i,,j)
2. **Character-string** data types are either fixed length-CHAR(n) or CHARACTER(n),where n is the number of characters-or varying length-VARCHAR(n) or CHAR VARYING(n) or CHARACTER VARYING(n),where n is maximum number of characters
3. **Bit-string** data types are either of fixed length n-BIT(n) or varying length BIT VARYING(n) where n is the maximum number of bits. The default for n, the length of a character string or bit string is 1
4. **A Boolean** data type has traditional values of TRUE or FALSE

5. **date and time** data type – the DATE data type has ten positions and its components are YEAR,MONTH and DAY in the form YYYY-MM-DD. The TIME data type has at least eight positions, with the components HOUR,MINUTE, and SECOND in the form HH:MM:SS

6. **A TIMESTAMP** data type includes DATE and TIME fields,plus a minimum of six positions for decimal fractions of seconds and an optional WITH TIME ZONE qualifier. For example, TIMESTAMP '2002-09-27 09:12:47 648302'

7. **INTERVAL** data type- Another data type related to DATE, TIME and TIMESTAMP is the INTERVAL data type. This specifies an interval- a relative value that can be used to increment or decrement an absolute value of a date ,time, or timestamp. Intervals are qualified to be either YEAR/MONTH intervals or DAY/TIME intervals

DBMS Languages:

DDL(table definitions)

Data Definition Language (DDL) statements are used to define the database structure or schema. 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
- RENAME - rename an object

DML

Data Manipulation Language (DML) statements are used for managing data within schema objects. 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
- MERGE - UPSERT operation (insert or update)
- CALL - call a PL/SQL or Java subprogram
- EXPLAIN PLAN - explain access path to data
- LOCK TABLE - control concurrency

DCL

Data Control Language (DCL) statements. Some examples:

- GRANT - gives user's access privileges to database
- REVOKE - withdraw access privileges given with the GRANT command

TCL

Transaction Control (TCL) statements are used to manage the changes made by DML statements. It allows statements to be grouped together into logical transactions.

- 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 isolation level and what rollback segment to use

DDL commands with examples

create command

create is a DDL command used to create a table or a database.

Creating a Database

To create a database in RDBMS, *create* command is used. Following is the Syntax,

```
create database database-name;
```

Example for Creating Database

```
create database Test;
```

The above command will create a database named **Test**.

Creating a Table

create command is also used to create a table. We can specify names and datatypes of various columns along.Following is the Syntax,

```
create table table-name
{
 column-name1 datatype1,
 column-name2 datatype2,
 column-name3 datatype3,
 column-name4 datatype4
};
```

create table command will tell the database system to create a new table with given table name and column information.

Example for creating Table

```
create table Student(id int, name varchar, age int);
```

The above command will create a new table **Student** in database system with 3 columns, namely id, name and age.

alter command

alter command is used for alteration of table structures. There are various uses of *alter* command, such as,

- to add a column to existing table
- to rename any existing column
- to change datatype of any column or to modify its size.
- *alter* is also used to drop a column.

To Add Column to existing Table

Using alter command we can add a column to an existing table. Following is the Syntax,

```
alter table table-name add(column-name datatype);
```

Here is an Example for this,

```
alter table Student add(address char);
```

The above command will add a new column *address* to the **Student** table

---

To Add Multiple Column to existing Table

Using alter command we can even add multiple columns to an existing table. Following is the Syntax,

```
alter table table-name add(column-name1 datatype1, column-name2 datatype2,
column-name3 datatype3);
```

Here is an Example for this,

```
alter table Student add(father-name varchar(60), mother-name varchar(60), dob
date);
```

The above command will add three new columns to the **Student** table

---

To Add column with Default Value

alter command can add a new column to an existing table with default values. Following is the Syntax,

```
alter table table-name add(column-name1 datatype1 default data);
```

Here is an Example for this,

```
alter table Student add(dob date default '1-Jan-99');
```

The above command will add a new column with default value to the **Student** table

---

To Modify an existing Column

alter command is used to modify data type of an existing column . Following is the Syntax,

```
alter table table-name modify(column-name datatype);
```

Here is an Example for this,

```
alter table Student modify(address varchar(30));
```

The above command will modify *address* column of the **Student table**

To Rename a column

Using alter command you can rename an existing column. Following is the Syntax,

```
alter table table-name rename old-column-name to column-name;
```

Here is an Example for this,

```
alter table Student rename address to Location;
```

The above command will rename *address* column to *Location*.

To Drop a Column

alter command is also used to drop columns also. Following is the Syntax,

```
alter table table-name drop(column-name);
```

Here is an Example for this,

```
alter table Student drop(address);
```

The above command will drop *address* column from the **Student table**

truncate command

truncate command removes all records from a table. But this command will not destroy the table's structure. When we apply truncate command on a table its Primary key is initialized. Following is its Syntax,

```
truncate table table-name
```

Here is an Example explaining it.

```
truncate table Student;
```

The above query will delete all the records of **Student** table.

truncate command is different from **delete** command. delete command will delete all the rows from a table whereas truncate command re-initializes a table(like a newly created table).

For eg. If you have a table with 10 rows and an auto_increment primary key, if you use *delete* command to delete all the rows, it will delete all the rows, but will not initialize the primary key, hence if you will insert any row after using delete command, the auto_increment primary key will start from 11. But in case of *truncate*command, primary key is re-initialized.

---

drop command

drop query completely removes a table from database. This command will also destroy the table structure. Following is its Syntax,

```
drop table table-name
```

Here is an Example explaining it.

```
drop table Student;
```

The above query will delete the **Student** table completely. It can also be used on Databases. For Example, to drop a database,

```
 drop database Test;
```

The above query will drop a database named **Test** from the system.

---

rename query

rename command is used to rename a table. Following is its Syntax,

```
rename table old-table-name to new-table-name
```

Here is an Example explaining it.

```
rename table Student to Student-record;
```

The above query will rename **Student** table to **Student-record**.

DML commands with examples

Data Manipulation Language (DML) statements are used for managing data in database. DML commands are not auto-committed. It means changes made by DML command are not permanent to database, it can be rolled back.

---

1) INSERT command

Insert command is used to insert data into a table. Following is its general syntax,

```
INSERT into table-name values(data1,data2,..)
```

Lets see an example,

Consider a table **Student** with following fields.

S_id	S_Name	age

```
INSERT into Student values(101,'Adam',15);
```

The above command will insert a record into **Student** table.

S_id	S_Name	age
101	Adam	15

Example to Insert NULL value to a column

Both the statements below will insert NULL value into **age** column of the Student table.

```
INSERT into Student(id,name) values(102,'Alex');
```

Or,

```
INSERT into Student values(102,'Alex',null);
```

The above command will insert only two column value other column is set to null.

S_id	S_Name	age
101	Adam	15
102	Alex	

Example to Insert Default value to a column

```
INSERT into Student values(103,'Chris',default)
```

S_id	S_Name	age
101	Adam	15
102	Alex	
103	chris	14

Suppose the **age** column of student table has default value of 14.

Also, if you run the below query, it will insert default value into the age column, whatever the default value may be.

```
INSERT into Student values(103,'Chris')
```

2) UPDATE command

Update command is used to update a row of a table. Following is its general syntax,

```
UPDATE table-name set column-name = value where condition;
```

Lets see an example,

```
update Student set age=18 where s_id=102;
```

S_id	S_Name	age
101	Adam	15
102	Alex	18

103	chris	14

Example to Update multiple columns

```
UPDATE Student set s_name='Abhi',age=17 where s_id=103;
```

The above command will update two columns of a record.

S_id	S_Name	age
101	Adam	15
102	Alex	18
103	Abhi	17

---

3) Delete command

Delete command is used to delete data from a table. Delete command can also be used with condition to delete a particular row. Following is its general syntax,

```
DELETE from table-name;
```

Example to Delete all Records from a Table

```
DELETE from Student;
```

The above command will delete all the records from **Student** table.

Example to Delete a particular Record from a Table

Consider the following **Student** table

S_id	S_Name	age
101	Adam	15
102	Alex	18
103	Abhi	17

```
DELETE from Student where s_id=103;
```

The above command will delete the record where s_id is 103 from **Student** table.

S_id	S_Name	age
101	Adam	15
102	Alex	18

DCL commands with examples

Data Control Language(DCL) is used to control privilege in Database. To perform any operation in the database, such as for creating tables, sequences or views we need privileges. Privileges are of two types,

- **System :** creating session, table etc are all types of system privilege.
- **Object :** any command or query to work on tables comes under object privilege.

DCL defines two commands,

- **Grant :** Gives user access privileges to database.
- **Revoke :** Take back permissions from user.

To Allow a User to create Session

```
grant create session to username;
```

To Allow a User to create Table

```
grant create table to username;
```

To provide User with some Space on Tablespace to store Table

```
alter user username quota unlimited on system;
```

To Grant all privilege to a User

```
grant sysdba to username
```

To Grant permission to Create any Table

```
grant create any table to username
```

To Grant permission to Drop any Table

```
grant drop any table to username
```

To take back Permissions

```
revoke create table from username
```

TCL commands with examples

Transaction Control Language(TCL) commands are used to manage transactions in database.These are used to manage the changes made by DML statements. It also allows statements to be grouped together into logical transactions.

Commit command

Commit command is used to permanently save any transaaction into database.

Following is Commit command's syntax,

```
commit;
```

Rollback command

This command restores the database to last commited state. It is also use with savepoint command to jump to a savepoint in a transaction.

Following is Rollback command's syntax,

```
rollback to savepoint-name;
```

Savepoint command

savepoint command is used to temporarily save a transaction so that you can rollback to that point whenever necessary.

Following is savepoint command's syntax,

```
savepoint savepoint-name;
```

Example of Savepoint and Rollback

Following is the **class** table,

ID	NAME
1	abhi
2	adam
4	alex

Lets use some SQL queries on the above table and see the results.

```
INSERT into class values(5,'Rahul');

commit;

UPDATE class set name='abhijit' where id='5';

savepoint A;

INSERT into class values(6,'Chris');

savepoint B;

INSERT into class values(7,'Bravo');

savepoint C;

SELECT * from class;
```

The resultant table will look like,

ID	NAME
1	abhi
2	adam
4	alex

5	abhijit
6	chris
7	bravo

Now **rollback** to **savepoint B**

```
rollback to B;

SELECT * from class;
```

The resultant table will look like

ID	**NAME**
1	abhi
2	adam
4	alex
5	abhijit
6	chris

Now **rollback** to **savepoint A**

```
rollback to A;

SELECT * from class;
```

The result table will look like

ID	**NAME**

1	abhi
2	adam
4	alex
5	abhijit

Basic SQL querying

SELECT query:

Select query is used to retrieve data from a tables. It is the most used SQL query. We can retrieve complete tables, or partial by mentioning conditions using WHERE clause.

Syntax of SELECT Query

```
SELECT column-name1, column-name2, column-name3, column-nameN from table-name;
```

Example for SELECT Query

Conside the following **Student** table,

S_id	S_Name	age	address
101	Adam	15	Noida
102	Alex	18	Delhi
103	Abhi	17	Rohtak

104	Ankit	22	Panipat

```
SELECT s_id, s_name, age from Student.
```

The above query will fetch information of s_id, s_name and age column from Student table

S_id	S_Name	age
101	Adam	15
102	Alex	18
103	Abhi	17
104	Ankit	22

Example to Select all Records from Table

A special character **asterisk** `*` is used to address all the data(belonging to all columns) in a query. *SELECT*statement uses `*` character to retrieve all records from a table.

```
SELECT * from student;
```

The above query will show all the records of Student table, that means it will show complete Student table as result.

S_id	S_Name	age	address
101	Adam	15	Noida
102	Alex	18	Delhi

103	Abhi	17	Rohtak
104	Ankit	22	Panipat

Example to Select particular Record based on Condition

```
SELECT * from Student WHERE s_name = 'Abhi';
```

103	Abhi	17	Rohtak

Example to Perform Simple Calculations using Select Query

Conside the following **Employee** table.

eid	**Name**	**age**	**salary**
101	Adam	26	5000
102	Ricky	42	8000
103	Abhi	22	10000
104	Rohan	35	5000

```
SELECT eid, name, salary+3000  from Employee;
```

The above command will display a new column in the result, showing 3000 added into existing salaries of the employees.

eid	Name	salary+3000
101	Adam	8000
102	Ricky	11000
103	Abhi	13000
104	Rohan	8000

Project query:

project query is used to create new relation by deleting columns from an existing relation i.e., A new relation is created from another existing relation by selecting only those columns requested by the user from projection. Following example (from above Employee table) show projection

SELECT age, eid

FROM EMPLOYEE;

using WHERE clause

Where clause is used to specify condition while retriving data from table. *Where* clause is used mostly with *Select*, *Update* and *Delete* query. If condititon specified by *where* clause is true then only the result from table is returned.

Syntax for WHERE clause

```
SELECT column-name1,
 column-name2,
 column-name3,
 column-nameN
```

```
from table-name WHERE [condition];
```

Example using WHERE clause

Consider a **Student** table,

s_id	s_Name	age	address
101	Adam	15	Noida
102	Alex	18	Delhi
103	Abhi	17	Rohtak
104	Ankit	22	Panipat

Now we will use a SELECT statement to display data of the table, based on a condition, which we will add to the SELECT query using WHERE clause.

```
SELECT s_id,
 s_name,
 age,
 address
from Student WHERE s_id=101;
```

s_id	s_Name	age	address
101	Adam	15	Noida

Text Fields vs. Numeric Fields

SQL requires single quotes around text values (most database systems will also allow double quotes).

However, numeric fields should not be enclosed in quotes:

Example

```
SELECT * FROM Customers
WHERE CustomerID=1;
```

Example

```
SELECT * FROM Customers
WHERE Country='Mexico';
```

Operators in The WHERE Clause

The following operators can be used in the WHERE clause:

Operator	Description
=	Equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern

IN	To specify multiple possible values for a column

What is an Operator in SQL?

An operator is a reserved word or a character used primarily in an SQL statement's WHERE clause to perform operation(s), such as comparisons and arithmetic operations.

Operators are used to specify conditions in an SQL statement and to serve as conjunctions for multiple conditions in a statement.

- Arithmetic operators
- Comparison operators
- Logical operators
- Operators used to negate conditions

SQL Arithmetic Operators:

Assume variable a holds 10 and variable b holds 20, then:

Show Examples

Operator	Description	Example
+	Addition - Adds values on either side of the operator	a + b will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	a - b will give -10
*	Multiplication - Multiplies values on either side of the operator	a * b will give 200

/	Division - Divides left hand operand by right hand operand	b / a will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	b % a will give 0

SQL Comparison Operators:

Assume variable a holds 10 and variable b holds 20, then:

Show Examples

Operator	Description	Example
=	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(a = b) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.
<>	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a <> b) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) is not true.
<=	Checks if the value of left operand is less than or equal to the value of	(a <= b)

	right operand, if yes then condition becomes true.	is true.
!<	Checks if the value of left operand is not less than the value of right operand, if yes then condition becomes true.	(a !< b) is false.
!>	Checks if the value of left operand is not greater than the value of right operand, if yes then condition becomes true.	(a !> b) is true.

SQL Logical Operators:

Here is a list of all the logical operators available in SQL.

Show Examples

Operator	Description
ALL	The ALL operator is used to compare a value to all values in another value set.
AND	The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
ANY	The ANY operator is used to compare a value to any applicable value in the list according to the condition.
BETWEEN	The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.
EXISTS	The EXISTS operator is used to search for the presence of a row in a specified table that meets certain criteria.
IN	The IN operator is used to compare a value to a list of literal values that have been specified.

LIKE	The LIKE operator is used to compare a value to similar values using wildcard operators.
NOT	The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. **This is a negate operator.**
OR	The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
IS NULL	The NULL operator is used to compare a value with a NULL value.
UNIQUE	The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

Numeric Functions: These are functions that accept numeric input and return numeric values.

Numeric functions are used to perform operations on numbers. They accept numeric values as input and return numeric values as output. Few of the Numeric functions are:

Function Name	Return Value
ABS (x)	Absolute value of the number '*x*'
CEIL (x)	Integer value that is Greater than or equal to the number '*x*'
FLOOR (x)	Integer value that is Less than or equal to the number '*x*'
TRUNC (x, y)	Truncates value of number '*x*' up to '*y*' decimal places
ROUND (x, y)	Rounded off value of the number '*x*' up to the number '*y*' decimal places

The following examples explains the usage of the above numeric functions

Function Name	Examples	Return Value
ABS (x)	ABS (1) ABS (-1)	1 -1
CEIL (x)	CEIL (2.83) CEIL (2.49) CEIL (-1.6)	3 3 -1
FLOOR (x)	FLOOR (2.83) FLOOR (2.49) FLOOR (-1.6)	2 2 -2
TRUNC (x, y)	ROUND (125.456, 1) ROUND (125.456, 0) ROUND (124.456, -1)	125.4 125 120
ROUND (x, y)	TRUNC (140.234, 2) TRUNC (-54, 1) TRUNC (5.7) TRUNC (142, -1)	140.23 54 5 140

These functions can be used on database columns.

For Example: Let's consider the product table used in sql joins. We can use ROUND to round off the unit_price to the nearest integer, if any product has prices in fraction.

```
SELECT ROUND (unit_price) FROM product;
```

2) Character or Text Functions:

Character or text functions are used to manipulate text strings. They accept strings or characters as input and can return both character and number values as output.

Few of the character or text functions are as given below:

Function Name	Return Value
LOWER (string_value)	All the letters in *'string_value'* is converted to lowercase.

UPPER (string_value)	All the letters in *'string_value'* is converted to uppercase.
INITCAP (string_value)	All the letters in *'string_value'* is converted to mixed case.
LTRIM (string_value, trim_text)	All occurrences of *'trim_text'* is removed from the left of *'string_value'*.
RTRIM (string_value, trim_text)	All occurrences of *'trim_text'* is removed from the right of *'string_value'* .
TRIM (trim_text FROM string_value)	All occurrences of *'trim_text'* from the left and right of *'string_value'* , *'trim_text'* can also be only one character long .
SUBSTR (string_value, m, n)	Returns *'n'* number of characters from *'string_value'* starting from the *'m'* position.
LENGTH (string_value)	Number of characters in *'string_value'* in returned.
LPAD (string_value, n, pad_value)	Returns *'string_value'* left-padded with *'pad_value'* . The length of the whole string will be of *'n'* characters.
RPAD (string_value, n, pad_value)	Returns *'string_value'* right-padded with *'pad_value'* . The length of the whole string will be of *'n'* characters.

For Example, we can use the above UPPER() text function with the column value as follows.

```
SELECT UPPER (product_name) FROM product;
```

The following examples explains the usage of the above character or text functions

Function Name	Examples	Return Value
LOWER(string_value)	LOWER('Good Morning')	good morning
UPPER(string_value)	UPPER('Good Morning')	GOOD MORNING
INITCAP(string_value)	INITCAP('GOOD MORNING')	Good Morning
LTRIM(string_value, trim_text)	LTRIM ('Good Morning', 'Good)	Morning
RTRIM (string_value, trim_text)	RTRIM ('Good Morning', ' Morning')	Good
TRIM (trim_text FROM string_value)	TRIM ('o' FROM 'Good Morning')	Gd Mrning

SUBSTR (string_value, m, n)	SUBSTR ('Good Morning', 6, 7)	Morning
LENGTH (string_value)	LENGTH ('Good Morning')	12
LPAD (string_value, n, pad_value)	LPAD ('Good', 6, '*')	**Good
RPAD (string_value, n, pad_value)	RPAD ('Good', 6, '*')	Good**

3) Date and time Functions:

These are functions that take values that are of datatype DATE as input and return values of datatypes DATE, except for the MONTHS_BETWEEN function, which returns a number as output.

Few date functions are as given below.

Function Name	Return Value
ADD_MONTHS (date, n)	Returns a date value after adding *'n'* months to the date *'x'*.
MONTHS_BETWEEN (x1, x2)	Returns the number of months between dates x1 and x2.
ROUND (x, date_format)	Returns the date *'x'* rounded off to the nearest century, year, month, date, hour, minute, or second as specified by the *'date_format'*.
TRUNC (x, date_format)	Returns the date *'x'* lesser than or equal to the nearest century, year, month, date, hour, minute, or second as specified by the 'date_format'.
NEXT_DAY (x, week_day)	Returns the next date of the *'week_day'* on or after the date *'x'* occurs.
LAST_DAY (x)	It is used to determine the number of days remaining in a month from the date *'x'*specified.
SYSDATE	Returns the systems current date and time.
NEW_TIME (x, zone1, zone2)	Returns the date and time in zone2 if date 'x' represents the time in zone1.

The below table provides the examples for the above functions

Function Name	Examples	Return Value

ADD_MONTHS ( )	ADD_MONTHS ('16-Sep-81', 3)	16-Dec-81
MONTHS_BETWEEN( )	MONTHS_BETWEEN ('16-Sep-81', '16-Dec-81')	3
NEXT_DAY( )	NEXT_DAY ('01-Jun-08', 'Wednesday')	04-JUN-08
LAST_DAY( )	LAST_DAY ('01-Jun-08')	30-Jun-08
NEW_TIME( )	NEW_TIME ('01-Jun-08', 'IST', 'EST')	31-May-08

4) Conversion Functions:

These are functions that help us to convert a value in one form to another form. For Ex: a null value into an actual value, or a value from one datatype to another datatype like NVL, TO_CHAR, TO_NUMBER, TO_DATE.

Few of the conversion functions available in oracle are:

Function Name	Return Value
TO_CHAR (x [,y])	Converts Numeric and Date values to a character string value. It cannot be used for calculations since it is a string value.
TO_DATE (x [, date_format])	Converts a valid Numeric and Character values to a Date value. Date is formatted to the format specified by *'date_format'*.
NVL (x, y)	If *'x'* is NULL, replace it with *'y'*. *'x'*and *'y'* must be of the same datatype.
DECODE (a, b, c, d, e, default_value)	Checks the value of *'a'*, if *a = b*, then returns *'c'*. If *a = d*, then returns *'e'*. Else, returns *default_value*.

The below table provides the examples for the above functions

Function Name	Examples	Return Value
TO_CHAR ()	TO_CHAR (3000, '$9999') TO_CHAR (SYSDATE, 'Day, Month YYYY')	$3000 Monday, June 2008
TO_DATE ()	TO_DATE ('01-Jun-08')	01-Jun-08
NVL ()	NVL (null, 1)	1

TO_CHAR function

TO_CHAR function is used to typecast a numeric or date input to character type with a format model (optional).

Syntax

```
TO_CHAR(number1, [format], [nls_parameter])
```

For number to character conversion, nls parameters can be used to specify decimal characters, group separator, local currency model, or international currency model. It is an optional specification - if not available, session level nls settings will be used. For date to character conversion, the nls parameter can be used to specify the day and month names, as applicable.

Dates can be formatted in multiple formats after converting to character types using TO_CHAR function. The TO_CHAR function is used to have Oracle 11g display dates in a particular format. Format models are case sensitive and must be enclosed within single quotes.

Consider the below SELECT query. The query format the HIRE_DATE and SALARY columns of EMPLOYEES table using TO_CHAR function.

```
SELECT first_name,
       TO_CHAR (hire_date, 'MONTH DD, YYYY') HIRE_DATE,
            TO_CHAR (salary, '$99999.99') Salary
FROM employees
WHERE rownum < 5;


FIRST_NAME           HIRE_DATE          SALARY
-------------------- ------------------ ----------
Steven               JUNE      17, 2003  $24000.00
Neena                SEPTEMBER 21, 2005  $17000.00
Lex                  JANUARY   13, 2001  $17000.00
Alexander            JANUARY   03, 2006   $9000.00
```

The first TO_CHAR is used to convert the hire date to the date format MONTH DD, YYYY i.e. month spelled out and padded with spaces, followed by the two-digit day of the month, and then the four-digit year. If you prefer displaying the month name in mixed case (that is, "December"), simply use this case in the format argument: ('Month DD, YYYY').

The second TO_CHAR function in Figure 10-39 is used to format the SALARY to display the currency sign and two decimal positions.

Oracle offers comprehensive set of format models. The below table shows the list of format models which can be used to typecast date and number values as character using TO_CHAR.

Format Model	Description
,(comma)	It returns a comma in the specified position. You can specify multiple commas in a number format model. Restrictions:A comma element cannot begin a number format model. A comma cannot appear to the right of a decimal character or period in a number format model.
.(period)	Returns a decimal point, which is a period (.) in the specified position. Restriction: You can specify only one period in a number format model
$	Returns value with a leading dollar sign
0	Returns leading zeros. Returns trailing zeros.
9	Returns value with the specified number of digits with a leading space if positive or with a leading minus if negative. Leading zeros are blank, except for a zero value, which returns a zero for the integer part of the fixed-point number.
B	Returns blanks for the integer part of a fixed-point number when the integer part is zero (regardless of "0"s in the format model).
C	Returns in the specified position the ISO currency symbol (the current value of the NLS_ISO_CURRENCY parameter).
D	Returns in the specified position the decimal character, which is the current value of the NLS_NUMERIC_CHARACTER parameter. The default is a period (.). Restriction: You can specify only one decimal character in a number format model.

EEE	Returns a value using in scientific notation.
FM	Returns a value with no leading or trailing blanks.
G	Returns in the specified position the group separator (the current value of the NLS_NUMERIC_CHARACTER parameter). You can specify multiple group separators in a number format model. Restriction: A group separator cannot appear to the right of a decimal character or period in a number format model
L	Returns in the specified position the local currency symbol (the current value of the NLS_CURRENCY parameter).
MI	Returns negative value with a trailing minus sign (-). Returns positive value with a trailing blank. Restriction: The MI format element can appear only in the last position of a number format model.
PR	Returns negative value in . It can appear only in the end of a number format model.
RN,rm	Returns a value as Roman numerals in uppercase. Returns a value as Roman numerals in lowercase. Value can be an integer between 1 and 3999.
S	Returns negative value with a leading or trailing minus sign (-). Returns positive value with a leading or trailing plus sign (+). Restriction: The S format element can appear only in the first or last position of a number format model.
TM	"Text minimum". Returns (in decimal output) the smallest number of characters possible. This element is case-insensitive.
U	Returns in the specified position the "Euro" (or other) dual currency symbol (the current value of the NLS_DUAL_CURRENCY parameter).

V	Returns a value multiplied by 10n (and if necessary, round it up), where n is the number of 9's after the "V".
X	Returns the hexadecimal value of the specified number of digits.

TO_NUMBER function

The TO_NUMBER function converts a character value to a numeric datatype. If the string being converted contains nonnumeric characters, the function returns an error.

Syntax

```
TO_NUMBER (string1, [format], [nls_parameter])
```

The below table shows the list of format models which can be used to typecast character values as number using TO_NUMBER.

Format Model	Description
CC	Century
SCC	Century BC prefixed with -
YYYY	Year with 4 numbers
SYYY	Year BC prefixed with -
IYYY	ISO Year with 4 numbers
YY	Year with 2 numbers
RR	Year with 2 numbers with Y2k compatibility
YEAR	Year in characters

SYEAR	Year in characters, BC prefixed with -
BC	BC/AD Indicator
Q	Quarter in numbers (1,2,3,4)
MM	Month of year 01, 02...12
MONTH	Month in characters (i.e. January)
MON	JAN, FEB
WW	Week number (i.e. 1)
W	Week number of the month (i.e. 5)
IW	Week number of the year in ISO standard.
DDD	Day of year in numbers (i.e. 365)
DD	Day of the month in numbers (i.e. 28)
D	Day of week in numbers(i.e. 7)
DAY	Day of the week in characters (i.e. Monday)
FMDAY	Day of the week in characters (i.e. Monday)
DY	Day of the week in short character description (i.e. SUN)

J	Julian Day (number of days since January 1 4713 BC, where January 1 4713 BC is 1 in Oracle)
HH,H12	Hour number of the day (1-12)
HH24	Hour number of the day with 24Hours notation (0-23)
AM, PM	AM or PM
MI, SS	Number of minutes and seconds (i.e. 59) ,
SSSSS	Number of seconds this day.
DS	Short date format. Depends on NLS-settings. Use only with timestamp.
DL	Long date format. Depends on NLS-settings. Use only with timestamp.
E	Abbreviated era name. Valid only for calendars: Japanese Imperial, ROC Official, Thai Buddha.
EE	The full era name
FF	The fractional seconds. Use with timestamp.
FF1..FF9	The fractional seconds. Use with timestamp. The digit controls the number of decimal digits used for fractional seconds.
FM	Fill Mode: suppresses blanks in output from conversion
FX	Format Exact: requires exact pattern matching between data and format model.

IYY OR IY OR I	The last 3,2,1 digits of the ISO standard year. Output only
RM	The Roman numeral representation of the month (I .. XII)
RR	The last 2 digits of the year.
RRRR	The last 2 digits of the year when used for output. Accepts fout-digit years when used for input.
SP	Spelled format. Can appear of the end of a number element. The result is always in english. For example month 10 in format MMSP returns "ten"
SPTH	Spelled and ordinal format; 1 results in first.
TH	Converts a number to it's ordinal format. For example 1 becoms 1st.
TS	Short time format. Depends on NLS-settings. Use only with timestamp.
TZD	Abbreviated time zone name. ie PST.
TZH,TZM	Time zone hour/minute displacement.
TZR	Time zone region
X	Local radix character. In America this is a period (.)

The SELECT queries below accept numbers as character inputs and prints them following the format specifier.

```
SELECT  TO_NUMBER('121.23', '9G999D99')

FROM DUAL
```

```
TO_NUMBER('121.23','9G999D99')
------------------------------
                        121.23


SELECT  TO_NUMBER('1210.73', '9999.99')
FROM DUAL;


TO_NUMBER('1210.73','9999.99')
------------------------------
                       1210.73
```

TO_DATE function

The function takes character values as input and returns formatted date equivalent of the same. The TO_DATE function allows users to enter a date in any format, and then it converts the entry into the default format used by Oracle 11g.

Syntax:

```
TO_DATE( string1, [ format_mask ], [ nls_language ] )
```

A format_mask argument consists of a series of elements representing exactly what the data should look like and must be entered in single quotation marks.

Format Model	Description
YEAR	Year, spelled out
YYYY	4-digit year
YYY,YY,Y	Last 3, 2, or 1 digit(s) of year.
IYY,IY,I	Last 3, 2, or 1 digit(s) of ISO year.
IYYY	4-digit year based on the ISO standard

RRRR	Accepts a 2-digit year and returns a 4-digit year.
Q	Quarter of year (1, 2, 3, 4; JAN-MAR = 1).
MM	Month (01-12; JAN = 01).
MON	Abbreviated name of month.
MONTH	Name of month, padded with blanks to length of 9 characters.
RM	Roman numeral month (I-XII; JAN = I).
WW	Week of year (1-53) where week 1 starts on the first day of the year and continues to the seventh day of the year.
W	Week of month (1-5) where week 1 starts on the first day of the month and ends on the seventh.
IW	Week of year (1-52 or 1-53) based on the ISO standard.
D	Day of week (1-7).
DAY	Name of day.
DD	Day of month (1-31).
DDD	Day of year (1-366).
DY	Abbreviated name of day.

J	Julian day; the number of days since January 1, 4712 BC.
HH12	Hour of day (1-12).
HH24	Hour of day (0-23).
MI,SS	Minute (0-59).
SSSSS	Seconds past midnight (0-86399).
FF	Fractional seconds. Use a value from 1 to 9 after FF to indicate the number of digits in the fractional seconds. For example, 'FF4'.
AM,PM	Meridian indicator
AD,BC	AD, BC indicator
TZD	Daylight savings information. For example, 'PST'
TZH,TZM,TZR	Time zone hour/minute/region.

The following example converts a character string into a date:

```
SELECT TO_DATE('January 15, 1989, 11:00 A.M.',  'Month dd, YYYY, HH:MI A.M.',
'NLS_DATE_LANGUAGE = American')

FROM DUAL;


TO_DATE('

---------

15-JAN-89
```