SQL

* Oracle is developed in 1970 by oracle corp.
* It is designed in C language

Data 🡺 Anything having any information is called data.

Data base 🡺 collection data is called database.

database management system (DBMS)

Relational database management system

DIfference between DBMS and RDBMS

|  |  |
| --- | --- |
| DBMS   * Only database, no relationship between one db to another. * no ER . Entity relationship * Data redundancy * inconsistency * data accuracy is low | RDBMS   * Relationship available using primary key and foreign key * ER available * No redundancy * No inconsistency * data accuracy is more |

SQL : structure query language

Data type : data type will occupy the meomory of a particular column.

Number 🡺 integer value 0 to 32

char 🡺 2000 bytes

varchar 🡺 2000 bytes

varchr2 🡺 4000 bytes (takes specific memory)

date 🡺

LOB 🡺 2 gb

CLOB

BLOB

RAW

BFILE

Operators

* arithmetic operator
* logical operator
* sql operator

arithmetic operator 🡺 **+ - \* / \*\***

**logical operator 🡺 and , or, not**

**sql operator 🡺 like not like**

**in not in**

**is NULL is not null**

**between .. and not between .. and**

**select \* from emp where sal is null;**

**selecr \* from emp where sal is not null;**

**Integrity constraints**

**Oracle provides several ways of controlling what kind of data can be input into a table.**

when a constraint is applied to single column , that is known as column level constraint.

when a single constraint is applied to more that one column, then is called as table level constraint.

**Constraints are categorized as follows.**

* Domain integrity constraints
* Not null
* Check
* Entity integrity constraints
* Unique
* Primary key
* Referential integrity constraints
* Foreign key

Constraints are always attached to a column not a table.

We can add constraints in three ways.

* Column level 🡺 along with the column definition
* Table level 🡺 after the table definition
* Alter level 🡺 using alter command

While adding constraints you need not specify the name but the type only, oracle will internally name the constraints.

If you want to give a name to the constraint, you have to use the constraint clause.

While adding constraints you need not specify the name but the type only, oracle will internally name the constraint.

If you want to give a name to the constraint, you must use the constraint clause.

**NOT NULL**

* This is used to avoid null values.
* We can add this constraint in column level only.

Ex:

SQL> create table student(**no number(2) not null**, name varchar(10), marks number(3));

SQL> create table student(**no number(2) constraint nn not null**, name varchar(10), marks number(3));

**CHECK**

* This is used to insert the values based on specified condition.
* We can add this constraint in all three levels.

Ex:

COLUMN LEVEL

SQL> create table student(no number(2) , name varchar(10), **marks number(3) check (marks > 300));**

SQL> create table student(no number(2) , name varchar(10), marks number(3) constraint ch

check(marks > 300));

TABLE LEVEL

SQL> create table student(no number(2) , name varchar(10), marks number(3), **check (marks > 300));**

SQL> create table student(no number(2) , name varchar(10), marks number(3), **constraint**

**ch check(marks > 300));**

ALTER LEVEL

SQL> alter table student **add check(marks>300);**

SQL> alter table student **add constraint ch check(marks>300);**

**UNIQUE**

* This is used to avoid duplicates but it allow nulls.
* We can add this constraint in all three levels.

Ex:

**COLUMN LEVEL**

SQL> create table student(**student\_no number(2) unique**, name varchar(10), marks number(3));

SQL> create table student(no number(2) constraint un unique, name varchar(10), marks

number(3));

**TABLE LEVEL**

SQL> create table student(**student\_no** number(2) , name varchar(10), marks number(3), **unique(student\_no));**

SQL> create table student(no number(2) , name varchar(10), marks number(3), constraint

un unique(no));

**ALTER LEVEL**

SQL> alter table student **add unique(student\_no);**

SQL> alter table student add constraint un unique(no);

**PRIMARY KEY**

* This is used to avoid duplicates and nulls.
* This will work as combination of unique and not null.
* **primary key = unique + not null**
* Primary key always attached to the parent table.
* We can add this constraint in all three levels.
* primary key can be defined for a combination of column. When a primary key consists of multiple columns, then it is called as a **composite primary key.**
* EX of composite primary key : create table student ( college varchar2(4),

roll\_no number(4),

name varchar(10),

primary key (college,roll\_no) );

Ex: **COLUMN LEVEL**

SQL> create table student(**no number(2) primary key**, name varchar(10), marks number(3));

SQL> create table student(no number(2) constraint pk primary key, name varchar(10), marks number(3));

**TABLE LEVEL**

SQL> create table student(no number(2) , name varchar(10), marks number(3), **primary key(no));**

SQL> create table student(no number(2) , name varchar(10), marks number(3), constraint

pk primary key(no));

**ALTER LEVEL**

SQL> alter table student **add primary key(no);**

SQL> alter table student add constraint pk primary key(no);

FOREIGN KEY

* This is used to reference the parent table primary key column which allows duplicates.
* Foreign key always attached to the child table.
* We can add this constraint in table and alter levels only

Ex:

TABLE LEVEL

SQL> create table emp(empno number(2), ename varchar(10), deptno number(2),

primary key(empno), **foreign key(deptno) references dept(deptno));**

SQL> create table emp(empno number(2), ename varchar(10), deptno number(2),

constraint pk primary key(empno), constraint fk foreign key(deptno) references dept(deptno));

ALTER LEVEL

SQL> alter table emp add foreign key(deptno) references dept(deptno);

SQL> alter table emp add constraint fk foreign key(deptno) references dept(deptno);

* Once the primary key and foreign key relationship has been created then you cannot remove any parent record if the dependent child exists.

USING ON DELTE CASCADE

* By using this clause you can remove the parent record even after the child exists.
* Because whenever you remove parent record oracle automatically removes all its dependent records from child table, if this clause is present while creating foreign key constraint.

Ex:

TABLE LEVEL

SQL> create table emp(empno number(2), ename varchar(10), deptno number(2),

primary key(empno), **foreign key(deptno) references dept(deptno) on delete cascade**);

SQL> create table emp(empno number(2), ename varchar(10), deptno number(2),

constraint pk primary key(empno), constraint fk foreign key(deptno) references

dept(deptno) on delete cascade)

ALTER LEVEL

SQL> alter table emp add **foreign key(deptno) references dept(deptno) on delete cascade**;

SQL> alter table emp add constraint fk foreign key(deptno) references dept(deptno) on

delete cascade;

DEFERRABLE CONSTRAINTS

* Each constraint has two additional attributes to support deferred checking of constraints.

1. Deferred initially immediate
2. Deferred initially deferred

* Deferred initially immediate checks for constraint violation at the time of insert.
* Deferred initially deferred checks for constraint violation at the time of commit.

Ex:

SQL> create table student(no number(2), name varchar(10), marks number(3), constraint

un unique(no) deferred initially immediate);

SQL> create table student(no number(2), name varchar(10), marks number(3), constraint

un unique(no) deferred initially deferred);

SQL> alter table student add constraint un unique(no) deferrable initially deferred;

SQL> set constraints all immediate;

This will enable all the constraints violations at the time of inserting.

SQL> set constraints all deferred;

This will enable all the constraints violations at the time of commit.

**OPERATIONS WITH CONSTRAINTS**

Possible operations with constraints as follows.

* Enable
* Disable
* Enforce
* Drop

**ENABLE 🡺**

* This will enable the constraint. Before enable, the constraint will check the existing data

Ex:

SQL> alter table student enable constraint un;

**DISABLE 🡺**

* This will disable the constraint.

Ex:

SQL> alter table student enable constraint un;

**ENFORCE 🡺**

* This will enforce the constraint rather than enable for future inserts or updates.
* This will not check for existing data while enforcing data.

Ex:

SQL> alter table student enforce constraint un;

**DROP 🡺**

* This will remove the constraint.

Ex:

SQL> alter table student drop constraint un;

Once the table is dropped, constraints automatically will drop.

SQL is divided into the following

* Data Definition Language **(DDL)** 🡺 **create, alter, drop, truncate, rename**
* Data Manipulation Language **(DML)** 🡺 **insert, update, delete**
* Data Retrieval Language **(DRL)** 🡺 **select**
* Transaction Control Language **(TCL)** 🡺 **commit, rollback, save point**
* Data Control Language **(DCL)** 🡺 **grant, revoke**

**Data Definition Language (DDL)**

**Create 🡺**

* **table name must be starts with char**
* **table should not exceed more than 37 char**
* **should not contain special char other than \_ and $**

**Create table <table name> (col1 datatype1, col2 datatype2 …coln datatypen);**

Ex:

SQL> create table student (s\_no number (2), name varchar (10), marks number (3));

**alter** 🡺

* this can be used to add or remove columns and to modify the precision of the datatype.
* adding column

syntax:

alter table <*table\_name*> add <*col datatype*>;

ex:

sql> alter table student add sdob date;

* removing column

syntax:

**alter table <*table\_name*> drop <*col datatype*>;**

ex:

sql> alter table student drop column sdob;

* increasing or decreasing precision of a column

syntax:

**alter table <*table\_name*> modify <*col datatype*>;**

ex:

sql> alter table student modify marks number(5);

\* to decrease precision the column should be empty.

* making column unused

syntax:

**alter table <*table\_name*> set unused column <*col*>;**

ex:

sql> alter table student set unused column marks;

even though the column is unused still it will occupy memory.

* dropping unused columns

syntax:

**alter table <*table\_name*> drop unused columns;**

ex:

sql> alter table student drop unused columns;

\* **you can not drop individual unused columns of a table.**

* renaming column

syntax:

**alter table <*table\_name*> rename column <*old\_col\_name*> to <*new\_col\_name*>;**

ex:

sql> alter table student rename column marks to smarks;

**drop** 🡺

This will be used to drop the database object;

Syntax:

**drop table <*table\_name*>;**

Ex: SQL> drop table student;

**truncate** 🡺

* **This can be used to delete the entire table data permanently.**

Syntax:

**truncate table <*table\_name*>;**

Ex 🡺 SQL> truncate table student;

**rename** 🡺

This will be used to rename the database object;

Syntax: **rename <old\_*table\_name*> to <*new\_table\_name*>;**

Ex: SQL> rename student to stud;

Data Manipulation Language **(DML)**

**insert**

* This will be used to insert the records into table.
* We have two methods to insert.

1. **By value method**
2. **By address method**

**USING VALUE METHOD**

Syntax: **insert into <*table\_name*) values (*value1, value2, value3 …. Valuen*);**

Ex:

SQL> insert into student values (1, ’sudha’, 100);

SQL> insert into student values (2, ’saketh’, 200);

* To insert a new record again you have to type entire insert command, if there are lot of

Records this will be difficult.

* This will be avoided by using address method.

**USING ADDRESS METHOD**

Syntax**: insert into <*table\_name*) values *(&col1, &col2, &col3 …. &coln*);**

* This will prompt you for the values but for every insert you must use forward slash.

Ex:

SQL> insert into student values (&no, '&name', &marks);

Enter value for no: 1

Enter value for name: Jagan

Enter value for marks: 300

old 1: insert into student values(&no, '&name', &marks)

new 1: insert into student values(1, 'Jagan', 300)

SQL> /

Enter value for no: 2

Enter value for name: Naren

Enter value for marks: 400

old 1: insert into student values(&no, '&name', &marks)

new 1: insert into student values(2, 'Naren', 400)

**INSERTING DATA INTO SPECIFIED COLUMNS USING VALUE METHOD 🡺**

Syntax:

**insert into <*table\_name*)(*col1, col2, col3 … Coln*) values (*value1, value2, value3 ….***

***Valuen*);**

Ex:

SQL> insert into student (no, name) values (3, ’Ramesh’);

SQL> insert into student (no, name) values (4, ’Madhu’);

**INSERTING DATA INTO SPECIFIED COLUMNS USING ADDRESS METHOD 🡺**

Syntax:

**insert into <*table\_name*)(*col1, col2, col3 … coln*) values *(&col1, &col2, &col3 …. &coln*);**

This will prompt you for the values but for every insert you must use forward slash.

Ex:

SQL> insert into student (no, name) values (&no, '&name');

Enter value for no: 5

Enter value for name: Visu

old 1: insert into student (no, name) values(&no, '&name')

new 1: insert into student (no, name) values(5, 'Visu')

SQL> /

Enter value for no: 6

Enter value for name: Rattu

old 1: insert into student (no, name) values(&no, '&name')

new 1: insert into student (no, name) values(6, 'Rattu')

**Update 🡺**

This can be used to modify the table data.

Syntax: **Update <*table\_name*> set <*col1*> = value1, <*col2*> = value2 where <*condition*>;**

Ex:

SQL> **update student set marks = 500;**

If you are not specifying any condition this will update entire table.

SQL> update student set marks = 500 where no = 2;

SQL> **update student set marks = 500, name = 'Venu' where no = 1;**

**DELETE 🡺**

**This can be used to delete the table data temporarily.**

Syntax: **Delete <*table\_name*> where <*condition*>;**

Ex:

SQL> **delete student;**

If you are not specifying any condition this will delete entire table.

SQL> **delete student where no = 2;**

**Data Retrieval Langauage : SELECT**

Syntax:

**Select \* from <*table\_name*>;**  -- here \* indicates all columns

or

Select *col1, col2, … coln* from <*table\_name*>;

Ex:

SQL> **select \* from student;**

NO NAME MARKS

--- ------ --------

1 Sudha 100

2 Saketh 200

1 Jagan 300

2 Naren 400

3 Ramesh

4 Madhu

5 Visu

6 Rattu

SQL> **select no, name, marks from student;**

NO NAME MARKS

--- ------ --------

1 Sudha 100

2 Saketh 200

1 Jagan 300

2 Naren 400

3 Ramesh

4 Madhu

5 Visu

6 Rattu

**CONDITIONAL SELECTIONS AND OPERATORS**

We have two clauses used in this

* **Where**
* **Order by**

**USING WHERE**

Syntax:

**select \* from <*table\_name*> where <*condition*>;**

The following are the different types of operators used in where clause.

* Arithmetic operators
* Comparison operators
* **Arithmetic operators -- highest precedence**

+, -, \*, /

* **Comparison operators**

1. =, !=, >, <, >=, <=, <>
2. between, not between
3. in, not in
4. null, not null
5. like

* **Logical operators**

1. And
2. Or **-- lowest precedence**
3. not

**=, >, <, >=, <=, !=, <>**

Ex:

SQL> **select \* from student where no = 2;**

NO NAME MARKS

--- ------- ---------

2 Saketh 200

2 Naren 400

SQL> **select \* from student where no < 2;**

NO NAME MARKS

--- ------- ----------

1 Sudha 100

1 Jagan 300

SQL> **select \* from student where no > 2;**

NO NAME MARKS

--- ------- ----------

3 Ramesh

4 Madhu

5 Visu

6 Rattu

SQL> **select \* from student where no <= 2;**

NO NAME MARKS

--- ------- ----------

1 Sudha 100

2 Saketh 200

1 Jagan 300

2 Naren 400

SQL> **select \* from student where no >= 2;**

NO NAME MARKS

--- ------- ---------

2 Saketh 200

2 Naren 400

3 Ramesh

4 Madhu

5 Visu

6 Rattu

SQL> **select \* from student where no != 2;**

NO NAME MARKS

--- ------- ----------

1 Sudha 100

1 Jagan 300

3 Ramesh

4 Madhu

5 Visu

6 Rattu

SQL> **select \* from student where no <> 2;**

NO NAME MARKS

--- ------- ----------

1 Sudha 100

1 Jagan 300

3 Ramesh

4 Madhu

5 Visu

6 Rattu

**AND**

**This will gives the output when all the conditions become true.**

Syntax: **select \* from <*table name*> where <*condition1*> and <*condition2*> and .. <*conditionn*>;**

Ex:

SQL> select \* from student where no = 2 and marks >= 200;

NO NAME MARKS

--- ------- --------

2 Saketh 200

2 Naren 400

**OR**

**This will give the output when either of the conditions becomes true.**

Syntax: **select \* from <*table\_name*> where <*condition1*> and <*condition2*> or .. <*conditionn*>;**

Ex: SQL> select \* from student where no = 2 or marks >= 200;

NO NAME MARKS

--- ------- ---------

2 Saketh 200

1 Jagan 300

2 Naren 400

**BETWEEN**

**This will gives the output based on the column and its lower bound, upper bound**.

Syntax: select \* from <*table\_name*> where <*col*> between <*lower bound*> and <*upper bound*>;

Ex:

SQL> **select \* from student where marks between 200 and 400;**

NO NAME MARKS

--- ------- ---------

2 Saketh 200

1 Jagan 300

2 Naren 400

**NOT BETWEEN**

This will gives the output based on the column which values are not in its lower bound,

upperbound.

Syntax:

select \* from <*table\_name*> where <*col*> **not between <*lower bound*> and <*upper bound*>;**

Ex:

SQL> **select \* from student where marks not between 200 and 400;**

NO NAME MARKS

--- ------- ---------

1 Sudha 100

**IN**

This will gives the output based on the column and its list of values specified.

Syntax:

**select \* from <*table\_name*> where <*col*> in ( *value1, value2, value3 … valuen*);**

Ex: SQL> select \* from student where no in (1, 2, 3);

NO NAME MARKS

--- ------- ---------

1 Sudha 100

2 Saketh 200

1 Jagan 300

2 Naren 400

3 Ramesh

**NOT IN**

This will gives the output based on the column which values are not in the list of values

specified.

Syntax: select \* from <*table\_name*> where <*col*> not in ( *value1, value2, value3 … valuen*);

Ex: SQL> select \* from student where no not in (1, 2, 3);

NO NAME MARKS

--- ------- ---------

4 Madhu

5 Visu

6 Rattu

**NULL**

This will gives the output based on the null values in the specified column.

Syntax: select \* from <*table\_name*> where <*col*> is null;

Ex: SQL> **select \* from student where marks is null;**

NO NAME MARKS

--- ------- ---------

3 Ramesh

4 Madhu

5 Visu

6 Rattu

**NOT NULL**

This will gives the output based on the not null values in the specified column.

Syntax:

select \* from <*table\_name*> where <*col*> is not null;

Ex:

SQL> **select \* from student where marks is not null;**

NO NAME MARKS

--- ------- ---------

1 Sudha 100

2 Saketh 200

1 Jagan 300

2 Naren 400

**LIKE**

This will be used to search through the rows of database column based on the pattern you

specify.

Syntax: select \* from <*table\_name*> where <*col*> like <*pattern*>;

Ex:

i) This will give the rows whose marks are 100.

SQL> **select \* from student where marks like 100;**

NO NAME MARKS

--- ------- ---------

1 Sudha 100

ii) This will give the rows whose name start with ‘S’.

SQL> **select \* from student where name like 'S%';**

NO NAME MARKS

--- ------- ---------

1 Sudha 100

2 Saketh 200

iii) This will give the rows whose name ends with ‘h’.

SQL> **select \* from student where name like '%h';**

NO NAME MARKS

--- ------- ---------

2 Saketh 200

3 Ramesh

iV) **This will give the rows whose name’s second letter start with ‘a’.**

SQL> **select \* from student where name like '\_a%';**

NO NAME MARKS

--- ------- --------

2 Saketh 200

1 Jagan 300

2 Naren 400

3 Ramesh

4 Madhu

6 Rattu

V) **This will give the rows whose name’s third letter start with ‘d’.**

SQL> **select \* from student where name like '\_\_d%';**

NO NAME MARKS

--- ------- ---------

1 Sudha 100

4 Madhu

Vi) **This will give the rows whose name’s second letter start with ‘t’ from ending.**

SQL> **select \* from student where name like '%\_t%';**

NO NAME MARKS

--- ------- ---------

2 Saketh 200

6 Rattu

Vii) **This will give the rows whose name’s third letter start with ‘e’ from ending.**

SQL> **select \* from student where name like '%e\_\_%';**

NO NAME MARKS

--- ------- ---------

2 Saketh 200

3 Ramesh

Viii) **This will give the rows whose name cotains 2 a’s.**

SQL> select \* from student where name like '%a% a %';

NO NAME MARKS

--- ------- ----------

1 Jagan 300

\* You have to specify the patterns in *like* using underscore ( \_ ).

**ORDER BY**

This will be used to ordering the columns data (ascending or descending).

Syntax: Select \* from <*table\_name*> order by <*col*> desc;

* By default oracle will use ascending order.
* If you want output in descending order you have to use *desc* keyword after the column.

Ex:

SQL> **select \* from student order by no;**

NO NAME MARKS

--- ------- ---------

1 Sudha 100

1 Jagan 300

2 Saketh 200

2 Naren 400

3 Ramesh

4 Madhu

5 Visu

6 Rattu

SQL> **select \* from student order by no desc;**

NO NAME MARKS

--- ------- ---------

6 Rattu

5 Visu

4 Madhu

3 Ramesh

2 Saketh 200

2 Naren 400

1 Sudha 100

1 Jagan 300

**Transaction Control Language (TCL)**

**COMMIT**

* This will be used to save the work.
* Commit is of two types.

1. **Implicit**
2. **Explicit**

a) **IMPLICIT**

* This will be issued by oracle internally in two situations.
* When any DDL operation is performed.
* When you are exiting from SQL \* PLUS.

b) **EXPLICIT**

* This will be issued by the user.

Syntax:

Commit or commit work;

\* Whenever you committed then the transaction was completed.

**ROLLBACK**

* This will undo the operation.
* This will be applied in two methods.

1. **Upto previous commit**
2. **Upto previous rollback**

Syntax:

Roll or roll work;

Or

Rollback or rollback work;

\* **While process is going on, if suddenly power goes then oracle will rollback the transaction.**

**USING SAVEPOINT**

You can use savepoints to rollback portions of your current set of transactions.

Syntax:

Savepoint <*savepoint\_name*>;

Ex:

SQL> savepoint s1;

SQL> insert into student values(1, ‘a’, 100);

SQL> savepoint s2;

SQL> insert into student values(2, ‘b’, 200);

SQL> savepoint s3;

SQL> insert into student values(3, ‘c’, 300);

SQL> savepoint s4;

SQL> insert into student values(4, ‘d’, 400);

**Before rollback**

SQL> select \* from student;

NO NAME MARKS

--- ------- ----------

1 a 100

2 b 200

3 c 300

4 d 400

SQL> rollback to savepoint s3;

Or

SQL> rollback to s3;

This will rollback last two records.

SQL> select \* from student;

NO NAME MARKS

--- ------- ----------

1 a 100

2 b 200

**Data Control Language**

DCL commands are used to granting and revoking the permissions.

**GRANT**

This is used to grant the privileges to other users.

Syntax:

Grant <*privileges*> on <*object\_name*> to <*user\_name*> [with grant option];

Ex:

SQL> **grant select on student to sudha**; -- you can give individual privilege

SQL> **grant select, insert on student to sudha**; -- you can give set of privileges

SQL> **grant all on student to sudha**; -- you can give all privileges

The sudha user must use dot method to access the object.

SQL> select \* from saketh.student;

The sudha user can not grant permission on student table to other users. To get this type of

option use the following.

SQL> grant all on student to sudha with grant option;

Now sudha user also grant permissions on student table.

**REVOKE**

This is used to revoke the privileges from the users to which you granted the privileges.

Syntax:

Revoke <*privileges*> on <*object\_name*> from <*user\_name*>;

Ex:

SQL> **revoke select on student form sudha;** -- you can revoke individual privilege

SQL> **revoke select, insert on student from sudha;** -- you can revoke set of privileges

SQL> **revoke all on student from sudha;** -- you can revoke all privileges