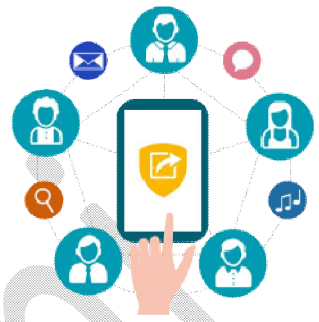


DATABASE MANAGEMENT SYSTEM II

L. J. COLLEGE OF COMPUTER APPLICATIONS

Prepared By | Parth Joshi

Database Management System II

Unit 1

Introduction to SQL

INDEX

1. Introduction to SQL

Structure Query Language (SQL) is a database query language used for storing and managing data in Relational DBMS. SQL was the first commercial language introduced for E.F Codd's **Relational** model of database.

SQL (*Structured Query Language*) is used to perform operations on the records stored in database such as updating records, deleting records, creating and modifying tables, views etc.

SQL is just a query language, it is not a database. To perform SQL queries, you need to install any database for example Oracle, MySQL, MongoDB, PostGre SQL, SQL Server, DB2 etc.

What is SQL

- SQL stands for **Structured Query Language**.
- It is designed for managing data in a relational database management system (RDBMS).
- It is pronounced as S-Q-L or sometime **See-Qwell**.
- SQL is a database language, it is used for database creation, deletion, fetching rows and modifying rows etc.
- SQL is based on relational algebra and tuple relational calculus.

All DBMS like MySQL, Oracle, MS Access, Sybase, Informix, Postgres and SQL Server use SQL as standard database language.

Why SQL is required

SQL is required:

- To create new databases, tables and views
- To insert records in a database
- To update records in a database
- To delete records from a database
- To retrieve data from a database

What SQL does

- With SQL, we can query our database in a numbers of ways, using English-like statements.
- With SQL, user can access data from relational database management system.
- It allows user to describe the data.
- It allows user to define the data in database and manipulate it when needed.
- It allows user to create and drop database and table.
- It allows user to create view, stored procedure, function in a database.
- It allows user to set permission on tables, procedure and view.

What is Database

A **database** is *an organized collection of data.*

Database handlers create database in such a way that only one set of software program provide access of data to all the users.

The **main purpose** of database is to operate large amount of information by storing, retrieving and managing.

There are many **dynamic websites** on the World Wide Web now days which are handled through databases. For example, a model to checks the availability of rooms in a hotel. It is an example of dynamic website that uses database.

There are many **databases available** like MySQL, Sybase, Oracle, Mango DB, Informix, Postgre, SQL Server etc.

SQL or Structured Query Language is used to perform operation on the data stored in a database. SQL depends on relational algebra and tuple relational calculus.

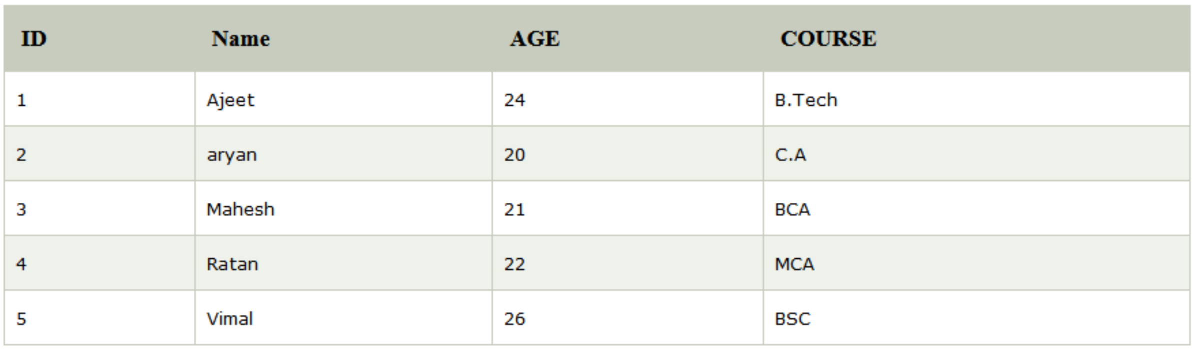
A cylindrical structure is used to display the image of a database.

What is table

The RDBMS database uses tables to store data. A table is a collection of related data entries and contains rows and columns to store data.

A table is the simplest example of data storage in RDBMS.

Let's see the example of student table.



ID	Name	AGE	COURSE
1	Ajeet	24	B.Tech
2	aryan	20	C.A
3	Mahesh	21	BCA
4	Ratan	22	MCA
5	Vimal	26	BSC

What is field?

Field is a smaller entity of the table which contains specific information about every record in the table. In the above example, the field in the student table consist of id, name, age, course.

What is row or record?

A row of a table is also called record. It contains the specific information of each individual entry in the table. It is a horizontal entity in the table. For example: The above table contains 5 records.

Let's see one record/row in the table.

1	Ajeet	24	B.Tech

What is column?

A column is a vertical entity in the table which contains all information associated with a specific field in a table. For example: "name" is a column in the above table which contains all information about student's name.

Ajeet
Aryan
Mahesh
Ratan
Vimal

NULL Values

The NULL value of the table specifies that the field has been left blank during record creation. It is totally different from the value filled with zero or a field that contains space.

Data Integrity

There are the following categories of data integrity exist with each RDBMS:

Entity integrity: It specifies that there should be no duplicate rows in a table.

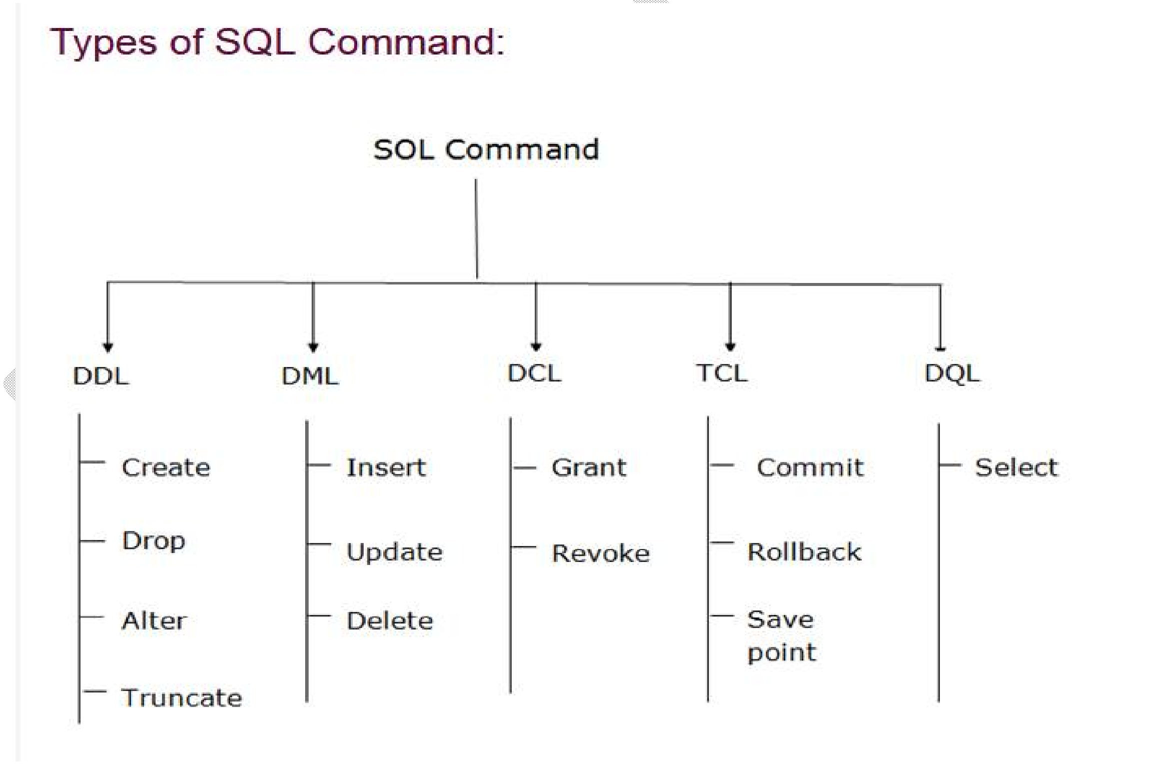
Domain integrity: It enforces valid entries for a given column by restricting the type, the format, or the range of values.

Referential integrity: It specifies that rows cannot be deleted, which are used by other records.

User-defined integrity: It enforces some specific business rules that are defined by users. These rules are different from entity, domain or referential integrity.

2. Data Definition Commands

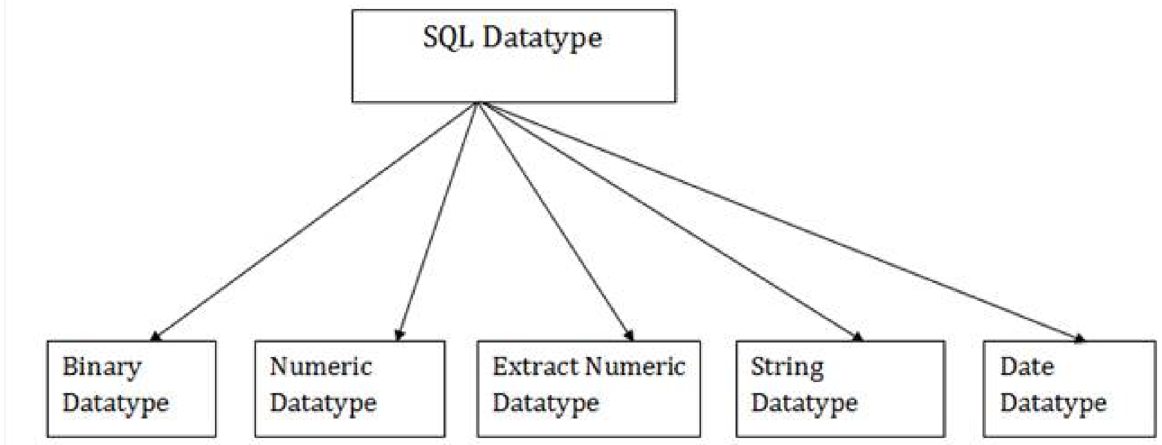
- SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.
- SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.



SQL Data type

- SQL Data type is used to define the values that a column can contain.
- Every column is required to have a name and data type in the database table.

Data type of SQL:



1. Binary Data types

There are Three types of binary Datatypes which are given below:

Data Type	Description
binary	It has a maximum length of 8000 bytes. It contains fixed-length binary data.
varbinary	It has a maximum length of 8000 bytes. It contains variable-length binary data.
image	It has a maximum length of 2,147,483,647 bytes. It contains variable-length binary data.

2. Approximate Numeric Datatype :

The subtypes are given below:

Data type	From	To	Description
float	-1.79E + 308	1.79E + 308	It is used to specify a floating-point value e.g. 6.2, 2.9 etc.
real	-3.40e + 38	3.40E + 38	It specifies a single precision floating point number

3. Exact Numeric Datatype

The subtypes are given below:

Data type	Description
int	It is used to specify an integer value.
smallint	It is used to specify small integer value.
bit	It has the number of bits to store.
decimal	It specifies a numeric value that can have a decimal number.
numeric	It is used to specify a numeric value.

4. Character String Datatype

The subtypes are given below:

Data type	Description
char	It has a maximum length of 8000 characters. It contains Fixed-length non-unicode characters.
varchar	It has a maximum length of 8000 characters. It contains variable-length non-unicode characters.
text	It has a maximum length of 2,147,483,647 characters. It contains variable-length non-unicode characters.

5. Date and time Datatypes

The subtypes are given below:

Datatype	Description
date	It is used to store the year, month, and days value.
time	It is used to store the hour, minute, and second values.
timestamp	It stores the year, month, day, hour, minute, and the second value.

1. Data definition language (DDL)

- DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
- All the command of DDL are auto-committed that means it permanently save all the changes in the database.

Here are some commands that come under DDL:

- CREATE
- ALTER
- DROP
- TRUNCATE

Creating a Table Structure

Create command can also be used to create tables. Now when we create a table, we have to specify the details of the columns of the tables too. We can specify the **names** and **datatypes** of various columns in the create command itself.

Syntax:

```
CREATE TABLE table_name (
column1 datatype [ NULL | NOT NULL ],
column2 datatype [ NULL | NOT NULL ],
column_n datatype [ NULL | NOT NULL ] );
```

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

Example for creating Table

```
CREATE TABLE Student(
   student_id NUMBER(20) NOT NULL,
   name VARCHAR2(100) NOT NULL,
   age NUMBER(5));
```

The above command will create a new table with name **Student** in the current database with 3 columns, namely `student_id`, `name` and `age`. Where the column `student_id` will only store integer, `name` will hold upto 100 characters and `age` will again store only integer value.

In Oracle, total number of columns cannot be more than 32.

SQL Constraints

SQL 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 the following two types,

1. **Column level constraints:** Limits only column data.
2. **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 this constraint is that it cannot be defined at table level.

Example using NOT NULL constraint

```
CREATE TABLE Student(
s_id number(10) NOT NULL,
Name varchar2(60),
Age number(5));
```

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. This constraint can be applied at column level or table level.

Using UNIQUE constraint when creating a Table (Table Level)

Here we have a simple CREATE query to create a table, which will have a column **s_id** with unique values.

CREATE TABLE Student(s_id number(10) NOT NULL UNIQUE, Name varchar2(60), Age number(5));

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

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.

Using PRIMARY KEY constraint at Table Level

CREATE table Student (s_id number(10) PRIMARY KEY, Name varchar2(60) NOT NULL, Age number(5));

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

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 its use, with help of the below tables:

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.

If you try to insert any incorrect data, DBMS will return error and will not allow you to insert the data.

Using FOREIGN KEY constraint at Table Level

```
CREATE table Order_Detail(
    order_id number(10) PRIMARY KEY,
    order_name varchar2(60) NOT NULL,
    c_id number(10) FOREIGN KEY REFERENCES Customer_Detail(c_id)
);
```

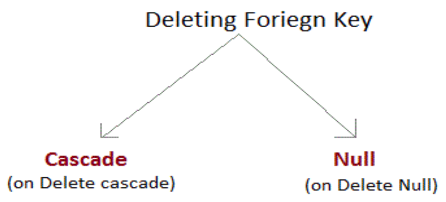
In this query, **c_id** in table Order_Detail is made as foriegn key, which is a reference of **c_id** column in Customer_Detail table.

Using FOREIGN KEY constraint at Column Level

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

Behaviour of Foreign Key Column on Delete

There are two ways to maintain the integrity of data in Child table, when a particular record is deleted in the main table. When two tables are connected with foreign key and certain data in the main table is deleted, for which a record exits in the child table, and then we must have some mechanism to save the integrity of data in the child table.



1. **On Delete Cascade:** This will remove the record from child table, if that value of foriegn key is deleted from the main table.
2. **On Delete Null:** This will set all the values in that record of child table as NULL, for which the value of foreign key is deleted from the main table.
3. 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.

Using CHECK constraint at Table Level

```
CREATE table Student(
   s_id number(10) NOT NULL CHECK(s_id > 0),
   Name varchar2(60) NOT NULL,
   Age number(5));
```

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

Using CHECK constraint at Column Level

```
ALTER table Student ADD CONSTRAINT check_id CHECK(s_id > 0);
```

3. Data Manipulation Commands

- DML commands are used to modify the database. It is responsible for all form of changes in the database.
- The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

Here are some commands that come under DML:

- INSERT
- UPDATE
- DELETE

- **Adding Table Rows**

INSERT command

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

INSERT INTO table_name VALUES(data1, data2, ...)

Let's see an example,

Consider a table **student** with the following fields.

s_id	name	age

INSERT INTO student VALUES(101, 'Bangad', 15);

The above command will insert a new record into **student** table.

s_id	name	age
101	Bangad	15

Insert value into only specific columns

We can use the INSERT command to insert values for only some specific columns of a row. We can specify the column names along with the values to be inserted like this,

INSERT INTO student(id, name) values(102, 'Katva');

The above SQL query will only insert id and name values in the newly inserted record.

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, 'Katva');

Or,

INSERT INTO Student VALUES(102,'Katva', null);

The above command will insert only two column values and the other column is set to null.

S_id	S_Name	age
101	Bangad	15
102	Katva	

Insert Default value to a column

INSERT INTO Student VALUES(103,'Gadani', default)

S_id	S_Name	age
101	Bangad	15
102	Katva	
103	Gadani	14

Suppose the column age in our table has a 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,'Gadani')

- **Saving Table Rows**

COMMIT Command

COMMIT command is used to permanently save any transaction into the database.

When we use any DML command like INSERT, UPDATE or DELETE, the changes made by these commands are not permanent, until the current session is closed, the changes made by these commands can be rolled back.

To avoid that, we use the COMMIT command to mark the changes as permanent.

Following is commit command's syntax,

COMMIT;

- **Listing Table Rows**

SELECT SQL Query

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

Syntax of SELECT query

SELECT query is used to retrieve records from a table. We can specify the names of the columns which we want in the result set.

```
SELECT
   column_name1,
   column_name2,
   column_name3,
   ...
   column_nameN
   FROM table_name;
```

Let us see an Example

Consider the following **student** table,

s_id	name	age	address
101	Adam	15	Chennai
102	Alex	18	Delhi
103	Abhi	17	Banglore
104	Ankit	22	Mumbai

SELECT s_id, name, age FROM student;

The above query will fetch information of s_id, name and age columns of the **student** table and display them,

s_id	name	age
101	Adam	15
102	Alex	18
103	Abhi	17
104	Ankit	22

As you can see the data from address column is absent, because we did not specif it in our SELECT query.

Select all records from a 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, for all the columns.

SELECT * FROM student;

The above query will show all the records of **student** table, that means it will show complete dataset of the table.

s_id	name	age	address
101	Adam	15	Chennai
102	Alex	18	Delhi
103	Abhi	17	Banglore
104	Ankit	22	Mumbai

- **Updating Table Rows**

SQL Update Statement

The SQL UPDATE statement is used to modify the data that is already in the database. The condition in the WHERE clause decides that which row is to be updated.

WHERE is used to add a condition to any SQL query, we will soon study about it in detail.

Syntax

UPDATE table_name SET column1 = value1, column2 = value2,.. WHERE condition;

Sample Table

EMPLOYEE

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Prashant	Ahmedabad	200000	30
2	Jaimin	Surat	300000	26
3	Raj	Baroda	100000	42
4	Muskan	Ahmedabad	500000	29
5	Dhruvil	Bhavnagar	200000	36
6	Umang	Surat	600000	48

Updating single record

Update the column EMP_NAME and set the value to 'Shivam' in the row where SALARY is 500000.

Syntax

UPDATE table_name SET column_name = value WHERE condition;

Query

```
UPDATE EMPLOYEE

        SET EMP_NAME = 'Shivam'

        WHERE SALARY = 500000;
```

Output: After executing this query, the EMPLOYEE table will look like:

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Prashant	Ahmedabad	200000	30
2	Jaimin	Surat	300000	26
3	Raj	Baroda	100000	42
4	Shivam	Ahmedabad	500000	29
5	Dhruvil	Bhavnagar	200000	36
6	Umang	Surat	600000	48

Updating multiple records

If you want to update multiple columns, you should separate each field assigned with a comma. In the EMPLOYEE table, update the column EMP_NAME to 'Smit' and CITY to 'Jaipur' where EMP_ID is 5.

Syntax

```
UPDATE table_name
        SET column_name = value1, column_name2 = value2
        WHERE condition;
```

Query

```
UPDATE EMPLOYEE
        SET EMP_NAME = 'Smit', City = 'Jaipur'
        WHERE EMP_ID = 5;
```

Output

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Prashant	Ahmedabad	200000	30
2	Jaimin	Surat	300000	26
3	Raj	Baroda	100000	42
4	Shivam	Ahmedabad	500000	29
5	Smitl	Jaipur	200000	36
6	Umang	Surat	600000	48

Without use of WHERE clause

If you want to update all row from a table, then you don't need to use the WHERE clause. In the EMPLOYEE table, update the column EMP_NAME as 'Ravi Kant'.

Syntax

UPDATE table_name

SET column_name = value1;

Query

UPDATE EMPLOYEE

SET EMP_NAME = 'Ravi Kant';

Output

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Ravi Kant	Ahmedabad	200000	30
2	Ravi Kant	Surat	300000	26
3	Ravi Kant	Baroda	100000	42
4	Ravi Kant	Ahmedabad	500000	29
5	Ravi Kant	Bhavnagar	200000	36
6	Ravi Kant	Surat	600000	48

- **Restoring Table Contents**

ROLLBACK command

This command restores the database to last committed state. It is also used with SAVEPOINT command to jump to a save point in an ongoing transaction.

If we have used the UPDATE command to make some changes into the database, and realise that those changes were not required, then we can use the ROLLBACK command to rollback those changes, if they were not committed using the COMMIT command.

Following is rollback command's syntax,

ROLLBACK TO savepoint_name; or ROLLBACK;

• Deleting Table Rows

SQL DELETE ROWS

The DELETE statement is used to delete rows from a table. If you want to remove a specific row from a table you should use WHERE condition.

DELETE FROM table_name [WHERE condition];

But if you do not specify the WHERE condition it will remove all the rows from the table.

DELETE FROM table_name;

There are some more terms similar to DELETE statement like as DROP statement and TRUNCATE statement but they are not exactly same there are some differences between them.

Difference between DELETE and TRUNCATE statements

There is a slight difference b/w delete and truncate statement. The **DELETE statement** only deletes the rows from the table based on the condition defined by WHERE clause or delete all the rows from the table when condition is not specified.

But it does not free the space containing by the table.

The **TRUNCATE statement:** it is used to delete all the rows from the table **and free the containing space.**

Let's see an "employee" table.

Emp_id	Name	Address	Salary
1	Aryan	Allahabad	22000
2	Shurabhi	Varanasi	13000
3	Khushboo	Delhi	24000

Execute the following query to truncate the table:

TRUNCATE TABLE employee;

Difference b/w DROP and TRUNCATE statements

When you use the drop statement it deletes the table's row together with the table's definition so all the relationships of that table with other tables will no longer be valid.

When you drop a table:

- Table structure will be dropped
- Relationship will be dropped
- Integrity constraints will be dropped
- Access privileges will also be dropped

On the other hand when we **TRUNCATE** a table, the table structure remains the same, so you will not face any of the above problems.

3. Select Query

- **with Conditional Restrictions**

Select a particular record based on a condition

You can select partial table contents by placing restrictions on the rows to be included in the output. This is done by using the WHERE clause to add conditional restrictions to the SELECT statement. The following syntax enables you to specify which rows to select:

SELECT *column list* FROM *table name*
[WHERE *condition list*];

The SELECT statement retrieves all rows that match the specified condition(s)—also known as the *conditional criteria*—you specified in the WHERE clause. The *condition list* in the WHERE clause of the SELECT statement is represented by one or more conditional expressions, separated by logical operators. “The WHERE clause is optional”.

Example:

SELECT * FROM student WHERE name = 'Bangad';

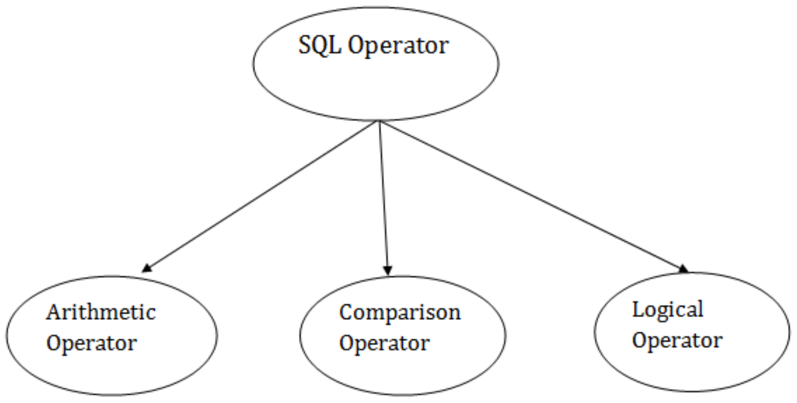
The above query will return the following result,

103	Bangad	17	Rohtak

If no rows match the specified criteria in the WHERE clause, you see a blank screen or a message that tells you that no rows were retrieved

• SQL Operators

There are various types of SQL operator:



SQL Arithmetic Operators

Let's assume 'variable a' and 'variable b'. Here, 'a' contains 20 and 'b' contains 10.

Operator	Description	Example
+	It adds the value of both operands.	a+b will give 30
-	It is used to subtract the right-hand operand from the left-hand operand.	a-b will give 10
*	It is used to multiply the value of both operands.	a*b will give 200
/	It is used to divide the left-hand operand by the right-hand operand.	a/b will give 2
%	It is used to divide the left-hand operand by the right-hand operand and returns reminder.	a%b will give 0

SQL Comparison Operators:

Let's assume 'variable a' and 'variable b'. Here, 'a' contains 20 and 'b' contains 10.

Operator	Description	Example
=	It checks if two operands values are equal or not, if the values are queal then condition becomes true.	(a=b) is not true
!=	It checks if two operands values are equal or not, if values are not equal, then condition becomes true.	(a!=b) is true
<>	It checks if two operands values are equal or not, if values are not equal then condition becomes true.	(a<>b) is true

>	It checks if the left operand value is greater than right operand value, if yes then condition becomes true.	(a>b) is not true
<	It checks if the left operand value is less than right operand value, if yes then condition becomes true.	(a<b) is true
>=	It checks if the left operand value is greater than or equal to the right operand value, if yes then condition becomes true.	(a>=b) is not true
<=	It checks if the left operand value is less than or equal to the right operand value, if yes then condition becomes true.	(a<=b) is true
!<	It checks if the left operand value is not less than the right operand value, if yes then condition becomes true.	(a!=b) is not true
!>	It checks if the left operand value is not greater than the right operand value, if yes then condition becomes true.	(a!>b) is true

SQL Logical and Special Operators

There is the list of logical operator used in SQL:

Operator	Description
ALL	It compares a value to all values in another value set.
AND	It allows the existence of multiple conditions in an SQL statement.
ANY	It compares the values in the list according to the condition.
BETWEEN	It is used to search for values that are within a set of values.
IN	It compares a value to that specified list value.
NOT	It reverses the meaning of any logical operator.
OR	It combines multiple conditions in SQL statements.
EXISTS	It is used to search for the presence of a row in a specified table.
LIKE	It compares a value to similar values using wildcard operator.

ORACLE SPECIAL OPERATORS

ANSI-standard SQL allows the use of special operators in conjunction with the WHERE clause. These special operators include:

BETWEEN:	Used to check whether an attribute value is within a range
IS NULL:	Used to check whether an attribute value is null
LIKE:	Used to check whether an attribute value matches a given string pattern
IN:	Used to check whether an attribute value matches any value within a value list
EXISTS:	Used to check whether a sub query returns any rows

The BETWEEN Special Operator

The operator BETWEEN may be used to check whether an attribute value is within a range of values. For example, if you want to see a listing for all products whose prices are between Rs. 50 and Rs. 100, use the following command sequence:

```
SELECT *
FROM PRODUCT
WHERE P_PRICE BETWEEN 50.00 AND 100.00;
```

The IS NULL Special Operator

Standard SQL allows the use of IS NULL to check for a null attribute value. For example, suppose that you want to list all products that do not have a vendor assigned (V_CODE is null). Such a null entry could be found by using the command sequence:

```
SELECT P_CODE, P_DESCRIPT, V_CODE
FROM PRODUCT
WHERE V_CODE IS NULL;
```

The LIKE Special Operator

The LIKE special operator is used in conjunction with wildcards to find patterns within string attributes. Standard SQL allows you to use the percent sign (%) and underscore (_) wildcard characters to make matches when the entire string is not known: _ % means any and all *following* or preceding characters are eligible.

For example,

'J%' includes Johnson, Jones, Jernigan, July, and J-231Q.
'Jo%' includes Johnson and Jones.
'%n' includes Johnson and Jernigan.

_ _ means any *one* character may be substituted for the underscore.

For example,

'_23-456-6789' includes 123-456-6789, 223-456-6789, and 323-456-6789.
'_23-_56-678_' includes 123-156-6781, 123-256-6782, and 823-956-6788.
'_o_es' includes Jones, Cones, Cokes, totes, and roles.

For example, the following query would find all VENDOR rows for contacts whose first names begin with *Smit*.

```
SELECT V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM VENDOR
WHERE V_CONTACT LIKE 'Smit%';
```

Keep in mind that most SQL implementations yield case-sensitive searches. For example, Oracle will not yield a result that includes *Jones* if you use the wildcard search delimiter 'jo%' in a search for last names. The reason is that *Jones* begins with a capital *J*, and your wildcard search starts with a lowercase *j*.

The IN Special Operator

Many queries that would require the use of the logical OR can be more easily handled with the help of the special operator IN. For example, the query:

```
SELECT *
FROM PRODUCT
WHERE V_CODE = 21344
OR V_CODE = 24288;
```

can be handled more efficiently with:

```
SELECT *
FROM PRODUCT
WHERE V_CODE IN (21344, 24288);
```

The EXISTS Special Operator

The EXISTS special operator can be used whenever there is a requirement to execute a command based on the result of another query. That is, if a subquery returns any rows, run the main query; otherwise, don't. For example, the following query will list all vendors, but only if there are products to order:

```
SELECT *
FROM VENDOR
WHERE EXISTS (SELECT * FROM PRODUCT WHERE P_QOH <= P_MIN);
```

ORACLE ALIASES

Alias is used to give an alias name to a table or a column, which can be a resultset table too. This is quite useful in case of large or complex queries. Alias is mainly used for giving a short alias name for a column or a table with complex names.

Syntax of Alias for table names,

```
SELECT column-name FROM table-name AS alias-name
```

Following is an SQL query using alias,

```
SELECT * FROM Employee_detail AS ed;
```

Syntax for defining alias for columns will be like,

SELECT column-name AS alias-name FROM table-name;

Example using alias for columns,

SELECT customer_id AS cid FROM Emp;

Example of Alias in SQL Query

Consider the following two tables,

The **class** table,

ID	Name
1	abhi
2	adam
3	alex
4	anu
5	ashish

and the **class_info** table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI
7	NOIDA
8	PANIPAT

Below is the Query to fetch data from both the tables using SQL Alias,

SELECT C.id, C.Name, Ci.Address from Class AS C, Class_info AS Ci where C.id = Ci.id;

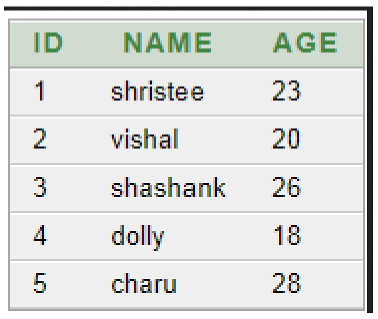
and the resultset table will look like,

ID	Name	Address
1	abhi	DELHI
2	adam	MUMBAI
3	alex	CHENNAI

SQL Alias seems to be quite a simple feature of SQL, but it is highly useful when you are working with more than 3 tables and have to use JOIN on them.

Some Examples of Select Query using Comparison, Arithmetic, Logical and Special Operators.

Table:



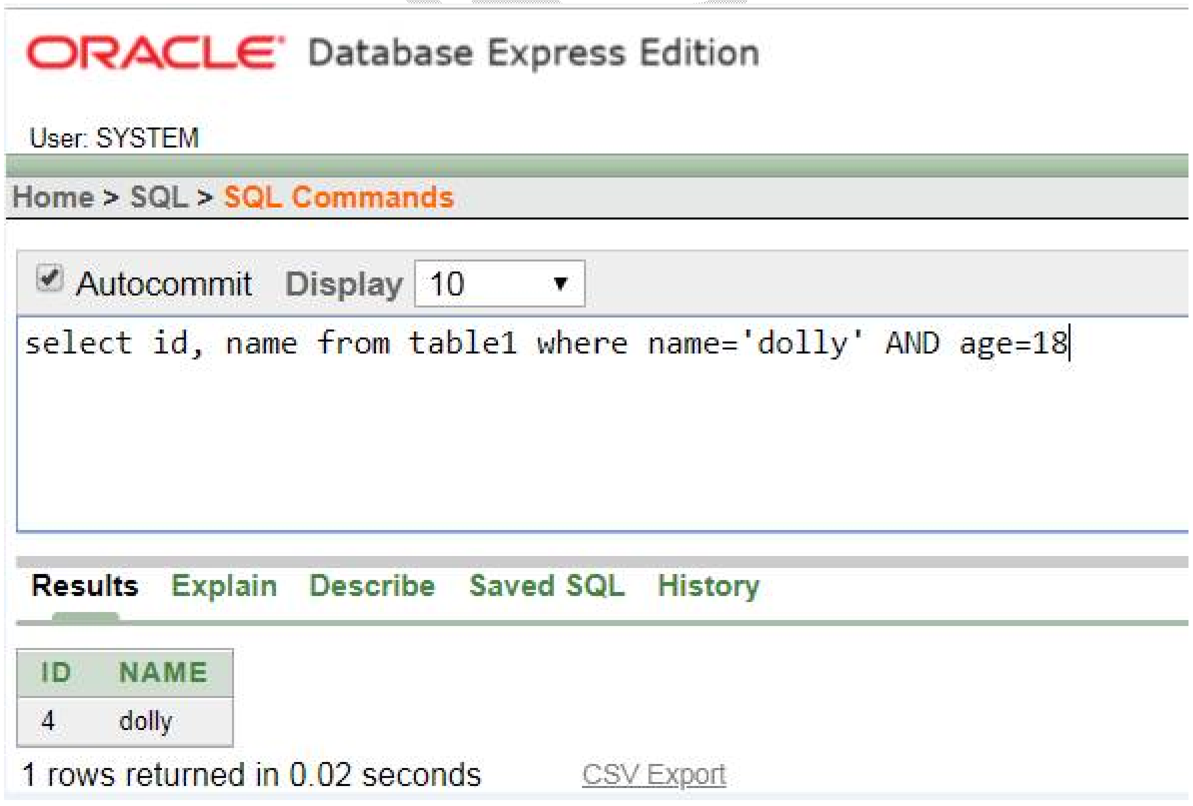
ID	NAME	AGE
1	shristee	23
2	vishal	20
3	shashank	26
4	dolly	18
5	charu	28

Example 1 (AND)

ORACLE AND

In Oracle, AND is used in select, insert, delete or update statement for checking two or more conditions.

select id, name from table1 where name='dolly' AND age=18



ORACLE Database Express Edition
User: SYSTEM
Home > SQL > SQL Commands
Autocommit Display 10
Results Explain Describe Saved SQL History
CSV Export

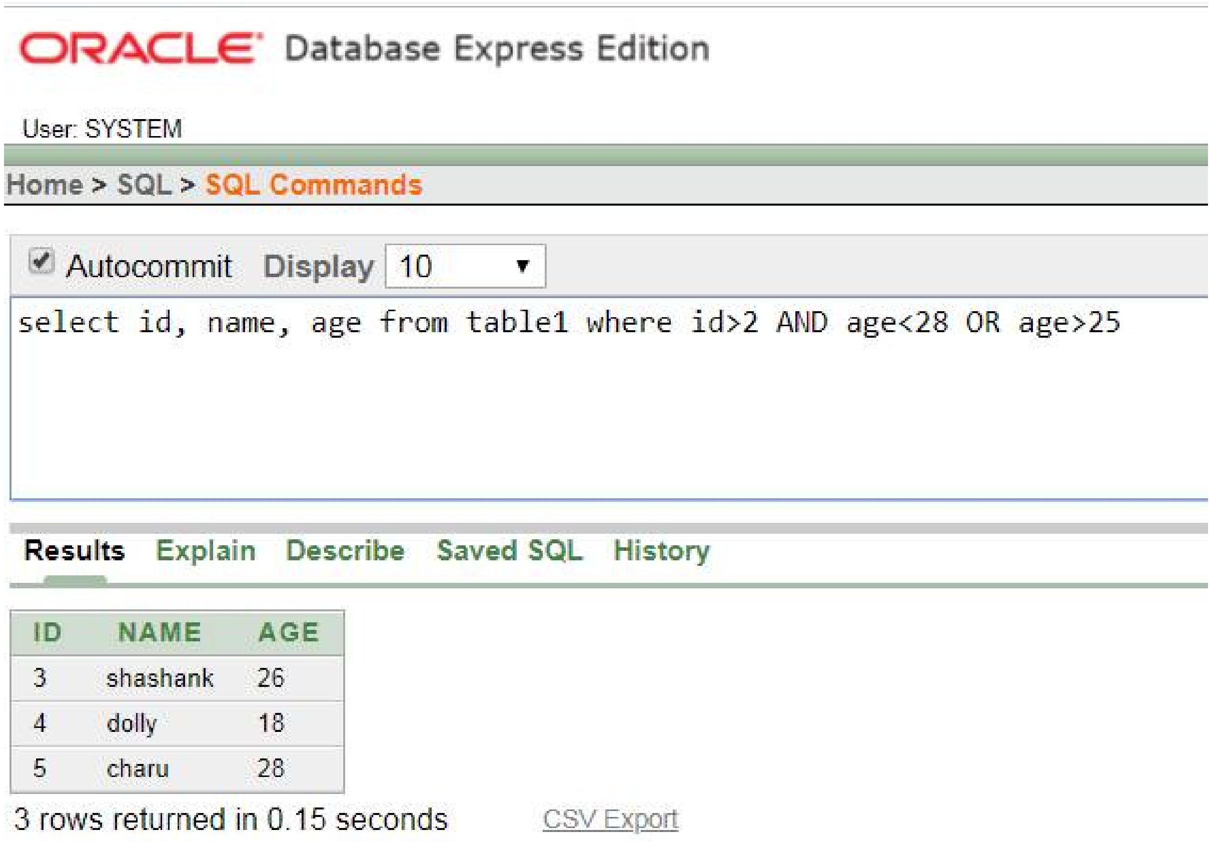
```
select id, name from table1 where name='dolly' AND age=18
```

ID	NAME
4	dolly

1 rows returned in 0.02 seconds

Example 2 (AND, OR)

select id, name, age from table1 where id>2 AND age<28 OR age>25



ORACLE Database Express Edition

User: SYSTEM

Home > SQL > SQL Commands

Autocommit Display 10

```
select id, name, age from table1 where id>2 AND age<28 OR age>25
```

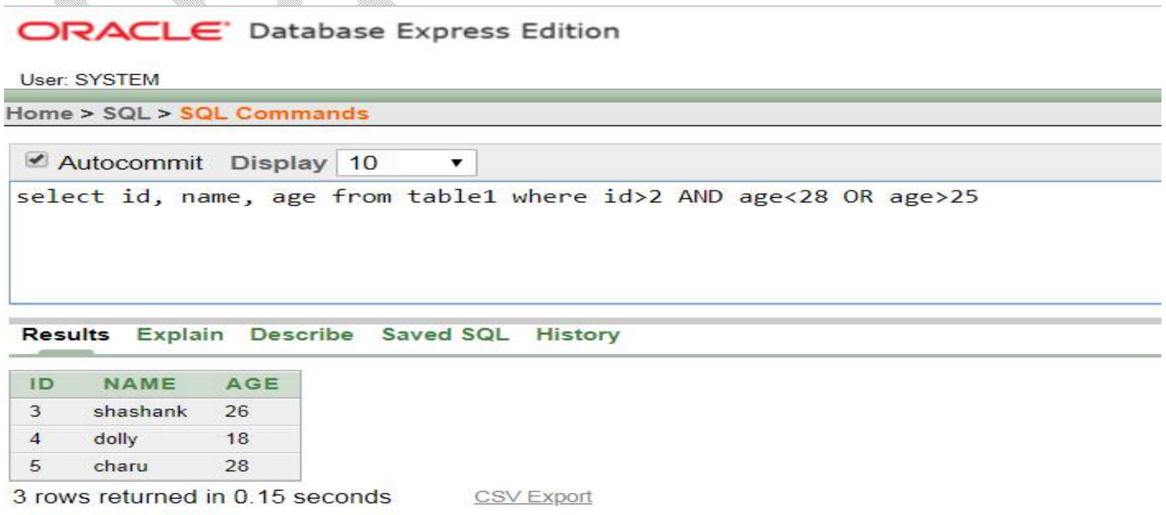
Results Explain Describe Saved SQL History

ID	NAME	AGE
3	shashank	26
4	dolly	18
5	charu	28

3 rows returned in 0.15 seconds CSV Export

Example 3 (BETWEEN)

select id, name, age from table1 where age between 20 AND 28



ORACLE Database Express Edition

User: SYSTEM

Home > SQL > SQL Commands

Autocommit Display 10

```
select id, name, age from table1 where id>2 AND age<28 OR age>25
```

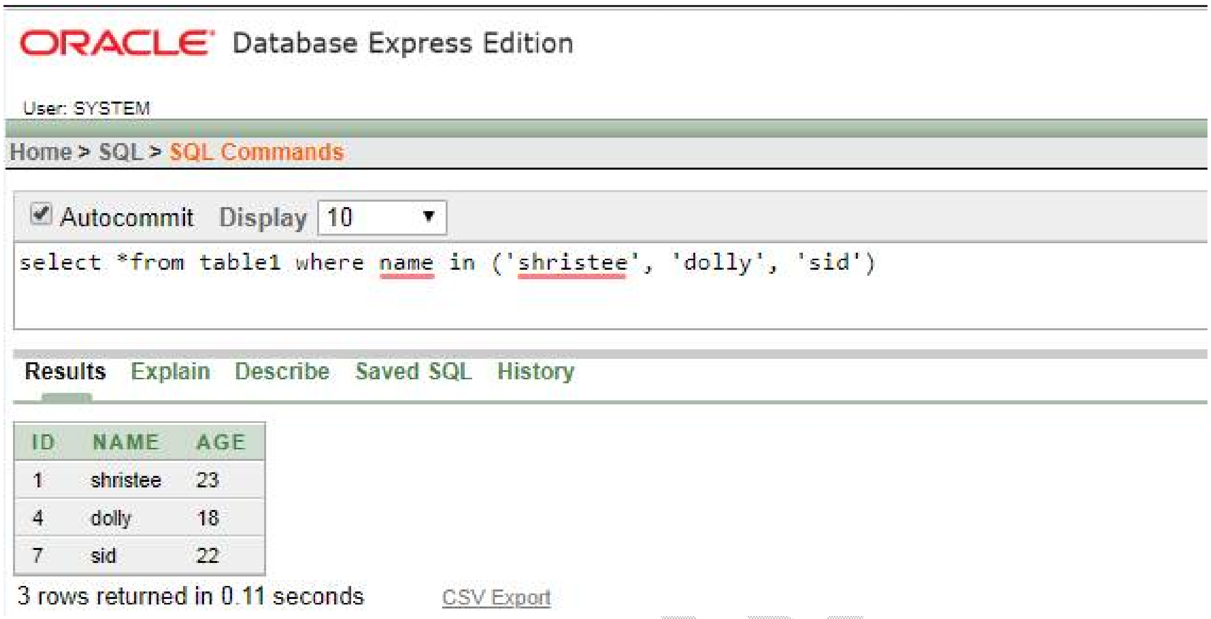
Results Explain Describe Saved SQL History

ID	NAME	AGE
3	shashank	26
4	dolly	18
5	charu	28

3 rows returned in 0.15 seconds CSV Export

Example 3 (IN)

select *from table1 where name in ('shristee', 'dolly', 'sid')


ORACLE Database Express Edition
User: SYSTEM
Home > SQL > SQL Commands
Autocommit Display 10
Results Explain Describe Saved SQL History
CSV Export

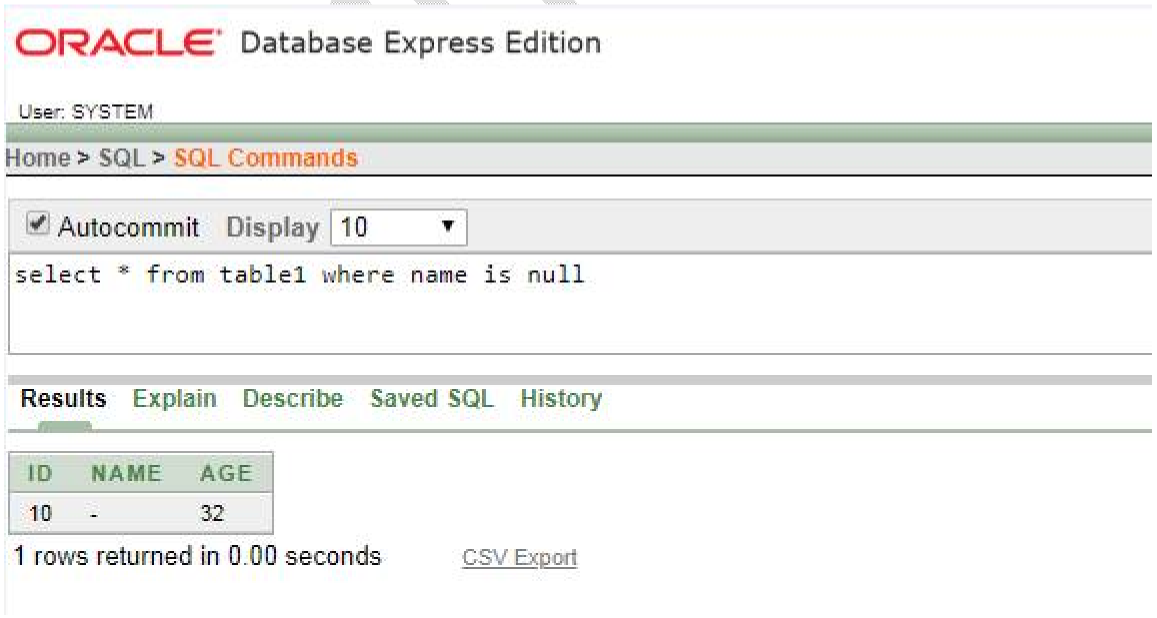
```
select *from table1 where name in ('shristee', 'dolly', 'sid')
```

ID	NAME	AGE
1	shristee	23
4	dolly	18
7	sid	22

3 rows returned in 0.11 seconds

Example 4 (IS NULL)

select * from table1 where name is null


ORACLE Database Express Edition
User: SYSTEM
Home > SQL > SQL Commands
Autocommit Display 10
Results Explain Describe Saved SQL History
CSV Export

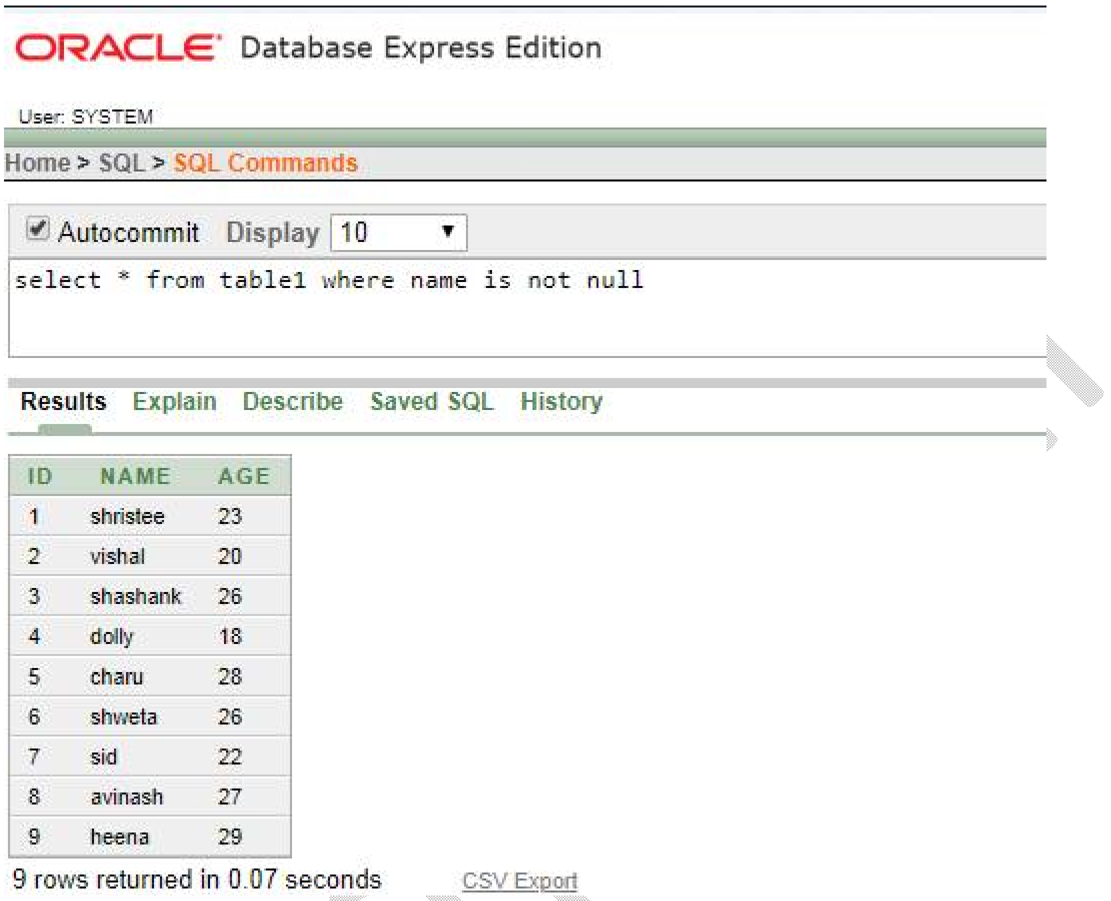
```
select * from table1 where name is null
```

ID	NAME	AGE
10	-	32

1 rows returned in 0.00 seconds

Example 5 (IS NOT NULL)

select * from table1 where name is not null


ORACLE Database Express Edition
User: SYSTEM
Home > SQL > SQL Commands
Autocommit Display 10
Results Explain Describe Saved SQL History
CSV Export

```
select * from table1 where name is not null
```

ID	NAME	AGE
1	shristee	23
2	vishal	20
3	shashank	26
4	dolly	18
5	charu	28
6	shweta	26
7	sid	22
8	avinash	27
9	heena	29

9 rows returned in 0.07 seconds

SQL LIKE clause

LIKE clause is used in the condition in SQL query with the WHERE clause. LIKE clause compares data with an expression using wildcard operators to match pattern given in the condition.

Wildcard operators

There are two wildcard operators that are used in LIKE clause.

- **Percent sign %**: represents zero, one or more than one character.
- **Underscore sign _**: represents only a single character.

Example of `LIKE` clause

Consider the following **Student** table.

s_id	s_Name	age
101	Adam	15
102	Alex	18
103	Abhi	17

SELECT * FROM Student WHERE s_name LIKE 'A%';

The above query will return all records where **s_name** starts with character 'A'.

s_id	s_Name	age
101	Adam	15
102	Alex	18
103	Abhi	17

Using _ and %

SELECT * FROM Student WHERE s_name LIKE '_d%';

The above query will return all records from **Student** table where **s_name** contain 'd' as second character.

s_id	s_Name	age
101	Adam	15

Using % only

SELECT * FROM Student WHERE s_name LIKE '%x';

The above query will return all records from **Student** table where **s_name** contain 'x' as last character.

s_id	s_Name	age
102	Alex	18

5. Advance Data Definition Commands

All changes in the table structure are made by using the **ALTER TABLE** command, followed by a keyword that produces the specific change you want to make. Three options are available: ADD, MODIFY, and DROP. You use ADD to add a column, MODIFY to change column characteristics, and DROP to delete a column from a table. Most RDBMSs do not allow you to delete a column (unless the column does not contain any values) because such an action might delete crucial data that are used by other tables. The basic syntax to add or modify columns is:

ALTER TABLE *tablename*
{ADD | MODIFY} (*columnname datatype* [{ADD | MODIFY} *columnname datatype*]) ;

The ALTER TABLE command can also be used to add table constraints. In those cases, the syntax would be:

ALTER TABLE *tablename*
ADD *constraint* [ADD *constraint*] ;

You could also use the ALTER TABLE command to remove a column or table constraint. The syntax would be as follows:

ALTER TABLE *tablename*
DROP{PRIMARY KEY | COLUMN *columnname* | CONSTRAINT *constraintname* };

Notice that when removing a constraint, you need to specify the name given to the constraint. That is one reason why you should always name your constraints in your CREATE TABLE or ALTER TABLE statement.

• Changing a Column's Data Type

Using the ALTER syntax, the (integer) V_CODE in the PRODUCT table can be changed to a character V_CODE by using:

ALTER TABLE PRODUCT
MODIFY (V_CODE CHAR(5));

• Changing a Column's Data Characteristics

If the column to be changed already contains data, you can make changes in the column's characteristics if those changes do not alter the data *type*. For example, if you want to increase the width of the P_PRICE column to nine digits, use the command:

```
ALTER TABLE PRODUCT
MODIFY (P_PRICE DECIMAL(9,2));
```

• Adding a Column

You can alter an existing table by adding one or more columns. In the following example, you add the column named P_SALECODE to the PRODUCT table.

```
ALTER TABLE PRODUCT
ADD (P_SALECODE CHAR(1));
```

When adding a column, be careful not to include the NOT NULL clause for the new column. Doing so will cause an error message; if you add a new column to a table that already has rows, the existing rows will default to a value of null for the new column. Therefore, it is not possible to add the NOT NULL clause for this new column.

• Dropping a Column

Occasionally, you might want to modify a table by deleting a column. Suppose that you want to delete the V_ORDER attribute from the VENDOR table. To accomplish that, you would use the following command:

```
ALTER TABLE VENDOR
DROP COLUMN V_ORDER;
```

• Adding Primary and Foreign Key Designations

When you create a new table based on another table, the new table does not include integrity rules from the old table. In particular, there is no primary key. To define the primary key for the new PART table, use the following command:

```
ALTER TABLE PART
ADD PRIMARY KEY (PART_CODE);
```

To Add Foreign Key Constraint following is the example:

```
ALTER TABLE PART
ADD FOREIGN KEY (V_CODE) REFERENCES VENDOR;
```

Add Multiple PRIMARY KEYS and FOREIGN KEYS following is the example:

```
ALTER TABLE LINE
ADD PRIMARY KEY (INV_NUMBER, LINE_NUMBER)
ADD FOREIGN KEY (INV_NUMBER) REFERENCES INVOICE
ADD FOREIGN KEY (PROD_CODE) REFERENCES PRODUCT;
```

• Deleting a Table from the Database

A table can be deleted from the database using the **DROP TABLE** command. For example, you can delete the PART table you just created with:

```
DROP TABLE PART;
```

Reference:

Database Systems
Design, Implementation and Management,
By: Peter Rob, Carlos Coronel