BIT 6113
Database Management System

Saroj Ghimire
saroj.ghimire2@texascollege.edu.np

Lincoln University College, CN 1221

SQL

Sequential query language

What is SQL

- SQL (Structured Query Language) is a standardized programming language that's used to manage relational databases and perform various operations on the data in them. Initially created in the 1970s, SQL is regularly used not only by database administrators, but also by developers writing data integration scripts and data analysts looking to set up and run analytical queries.
- The uses of SQL include modifying database table and index structures; adding, updating and deleting rows of data; and retrieving subsets of information from within a database for transaction processing and analytics applications. Queries and other SQL operations

take the form of commands written as statements -- commonly used SQL statements include select, add, insert, update, delete, create, alter and truncate.

- SQL became the de facto standard programming language for relational databases after they emerged in the late 1970s and early 1980s. Also known as SQL databases, relational systems comprise a set of tables containing data in rows and columns. Each column in a table corresponds to a category of data -- for example, customer name or address -- while each row contains a data value for the intersecting column.

Types Of SQL Commands : Introduction

- **SQL Categorizes its commands on the basis of functionalities performed by them. There are five types of SQL Commands which can be classified as:**
  - **DDL(Data Definition Language).**

- **DML(Data Manipulation Language).**
- **DQL(Data Query Language).**
- **DCL(Data Control Language).**
- **TCL(Transaction Control Language).**

Types Of SQL Commands : Data Definition Language(DDL)

• In order to make/perform changes on the physical structure of any table residing inside a database, DDL is used. These commands when executed are auto commit in nature and all the changes in the table are reflected and saved immediately. DDL commands includes :

S.No	DDL Commands	Description	Sample Query
1.	CREATE	Used to create tables or databases.	CREATE table student;
2.	ALTER	Used to modify the values in the tables.	ALTER table student add column roll_no int;
3.	RENAME	Used to rename the table or database name.	RENAME student to student_details;
4.	DROP	Deletes the table from the database.	DROP table student_details;
5.	TRUNCATE	Used to delete a table from database.	TRUNCATE table student_details;

Types Of SQL Commands : Data Manipulation Language(DML)

• Once the tables are created and database is generated using DDL commands, manipulation inside those tables and databases is done using DML commands. The advantage of using DML commands is, if in case any wrong changes or values are made, they can be changes and rolled back easily. DML commands includes:

S.No	DML Command	Description	Sample Query
1.	INSERT	Used to insert new rows in the tables.	INSERT into student(roll_no, name) values(1, Anoop);
2.	DELETE	Used to delete a row or entire table.	DELETE table student;
3.	UPDATE	Used to update values of existing rows of tables.	UPDATE students set s_name = 'Anurag' where s_name like 'Anoop';
4.	LOCK	Used to lock the privilege as either read or write.	LOCK tables student read;
5.	MERGE	Used to merge two rows of existing tables in database.	

Types Of SQL Commands : Data Control

Language(DCL)

- **DCL commands as the name suggests manages the matters and issues related to the data control in any database. TCL commands mainly provides special privilege access to users and is also used to specify the roles of users accordingly. There are two commonly used DCL commands, these are:**

S.No	DCL Commands	Description	Sample Query
1.	GRANT	Used to provide access to users.	GRANT CREATE table to user1;
2.	REVOKE	Used to take back the access privileges from the users.	REVOKE CREATE table from user1;

Types Of SQL Commands : Data Query

Language(DQL)

- **Data query language consists of only one command over which data selection in SQL relies. SELECT command in combination with other SQL clauses is used to retrieve and fetch data from database/tables on the basis of certain conditions applied by user.**

S.No	DQL Command	Description	Sample Query
1.	SELECT	Used to fetch data from tables/database.	SELECT * from student_details;

Types Of SQL Commands : Transaction Control Language(TCL)

S.No	TCL Commands	Description	Sample Query
1.	ROLL BACK	Used to cancel or UNDO the changes made in the database.	ROLLBACK;
2.	COMMIT	Used to deploy or apply or save the changes in the database.	COMMIT;
3.	SAVEPOINT	Used to save the data on temporary basis in the database.	SAVEPOINT roll_no;

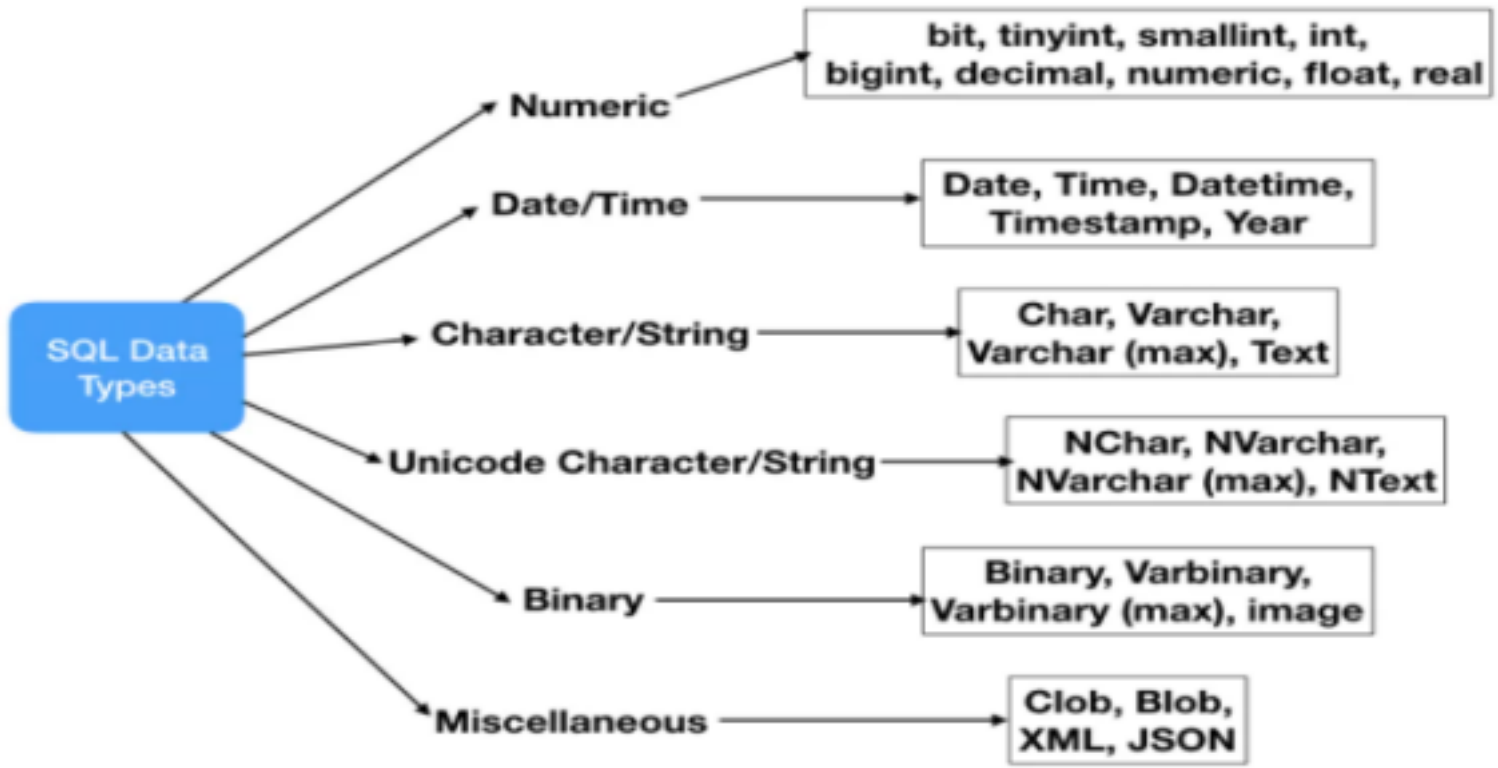
- **Transaction Control Language as the name suggests manages the issues and matters related to the transactions in any database. They are used to rollback or commit the changes in the database.**
- **Roll back means "Undo" the changes and Commit means "Applying" the changes. There are three major TCL commands.**

SQL Data Types

1.Numeric data types such as int, tinyint, bigint, float, real, etc.

2. Date and Time data types such as Date, Time, Datetime, etc.
3. Character and String data types such as char, varchar, text, etc.
4. Unicode character string data types, for example nchar, nvarchar, ntext, etc.
5. Binary data types such as binary, varbinary, etc.
6. Miscellaneous data types – clob, blob, xml, cursor, table, etc.

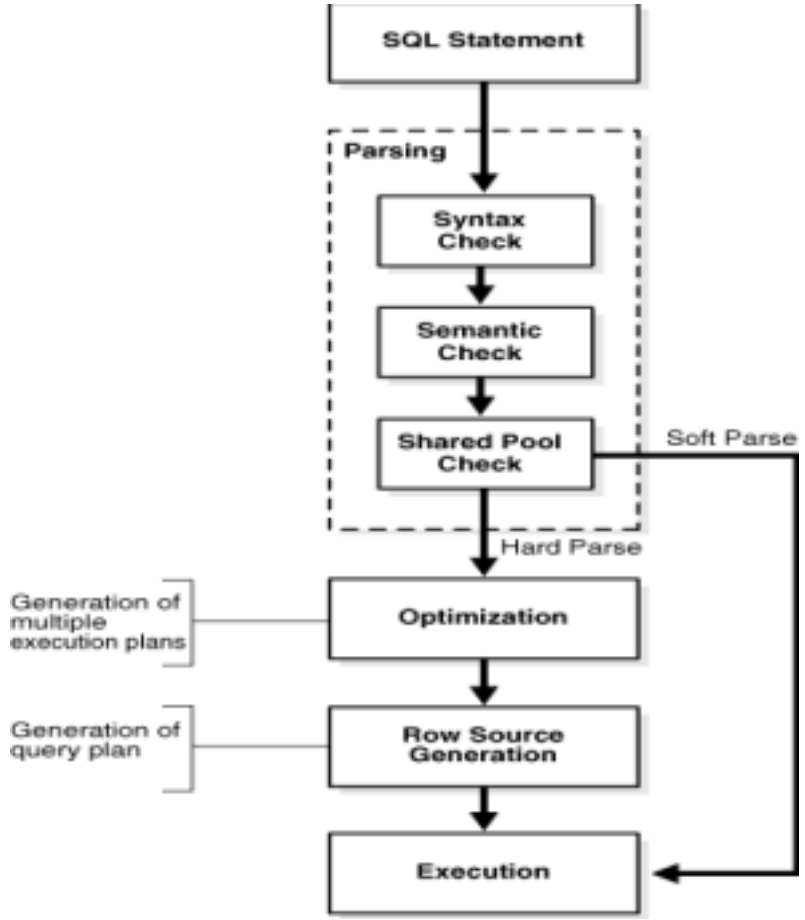
SQL Data Types


SQL Data Types
Numeric
bit, tinyint, smallint, int, bigint, decimal, numeric, float, real
Date/Time
Date, Time, Datetime, Timestamp, Year
Character/String
Char, Varchar, Varchar (max), Text
Unicode Character/String
NChar, NVarchar, NVarchar (max), NText
Binary
Binary, Varbinary, Varbinary (max), image
Miscellaneous
Clob, Blob, XML, JSON

SQL Data Types

Datatype	Description	Column Length and Default
CHAR (*size*)	Fixed-length character data of length *size* bytes.	Fixed for every row in the table (with trailing blanks); maximum size is 2000 bytes per row, default size is 1 byte per row. Consider the character set (one-byte or multibyte) before setting *size*.
VARCHAR2 (*size*)	Variable-length character data.	Variable for each row, up to 4000 bytes per row. Consider the character set (one-byte or multibyte) before setting *size*. A maximum *size* must be specified.
NCHAR(*size*)	Fixed-length character data of length *size* characters or bytes, depending on the national character set.	Fixed for every row in the table (with trailing blanks). Column *size* is the number of characters for a fixed-width national character set or the number of bytes for a varying-width national character set. Maximum *size* is determined by the number of bytes required to store one character, with an upper limit of 2000 bytes per row. Default is 1 character or 1 byte, depending on the character set.
NVARCHAR2 (*size*)	Variable-length character data of length *size* characters or bytes, depending on national character set. A maximum *size* must be specified.	Variable for each row. Column *size* is the number of characters for a fixed-width national character set or the number of bytes for a varying-width national character set. Maximum *size* is determined by the number of bytes required to store one character, with an upper limit of 4000 bytes per row. Default is 1 character or 1 byte, depending on the character set.
CLOB	Single-byte character data.	Up to 2^32 - 1 bytes, or 4 gigabytes.
NCLOB	Single-byte or fixed-length multibyte national character set (NCHAR) data.	Up to 2^32 - 1 bytes, or 4 gigabytes.
LONG	Variable-length character data.	Variable for each row in the table, up to 2^31 - 1 bytes, or 2 gigabytes, per row. Provided for backward compatibility.
NUMBER (p, s)	Variable-length numeric data. Maximum precision *p* and/or scale *s* is 38.	Variable for each row. The maximum space required for a given column is 21 bytes per row.
DATE	Fixed-length date and time data, ranging from Jan. 1, 4712 B.C.E. to Dec. 31, 4712 C.E.	Fixed at 7 bytes for each row in the table. Default format is a string (such as DD-MON-YY) specified by NLS_DATE_FORMAT parameter.
BLOB	Unstructured binary data.	Up to 2^32 - 1 bytes, or 4 gigabytes.
BFILE	Binary data stored in an external file.	Up to 2^32 - 1 bytes, or 4 gigabytes.
RAW (size)	Variable-length raw binary data.	Variable for each row in the table, up to 2000 bytes per row. A maximum *size* must be specified. Provided for backward compatibility.
LONG RAW	Variable-length raw binary data.	Variable for each row in the table, up to 2^31 - 1 bytes, or 2 gigabytes, per row. Provided for backward compatibility.
ROWID	Binary data representing row addresses.	Fixed at 10 bytes (extended ROWID) or 6 bytes (restricted ROWID) for each row in the table.
MLSLABEL	Trusted Oracle datatype.	See the *Trusted Oracle* documentation.

How Does A SQL Query Execute Inside Oracle? What Are The Steps Followed In The Background?



How Does A SQL Query Execute Inside Oracle? What Are The Steps Followed In The Background?

• Basically, there are 4-5 steps. There are two possible ways of query being executed. 1.
Soft Parse- Improves performance. This is also called as **'Library Cache Hit'**.
2. **Hard Parse**- Degrades performance. This is also called as **'Library Cache Miss'**.

Let us understand the steps one by one briefly.
There is a **parser** inside Oracle, which parses the SQL Query for further processing.
The first 3 steps of **SQL Query Execution** are taken care of by the parser.

• **Step 1: Syntax check**

This checks only for **syntax errors**. A pure syntax check and nothing else.
Examples: 1. Are the **keywords** correct?
Like, '**seelect', 'form', 'groupby', 'orderby'** are few common spelling mistakes.
2. Does SQL Query end with a **semicolon** (;)?
3. Are column names in the **'Select'** Clause are separated by **commas** (,)?
4. Does **'Select'** clause include only the colmuns which are present in the **'Group by'** clause? 5. Does **'Group by'** clause appear after **'Where clause'**?
6. Is **'Order by'** clause the **last clause** in the query?

etc. etc....

In each of these cases, if the answer is 'No', oracles throws an **error** stating the same.

How Does A SQL Query Execute Inside Oracle? What Are The Steps Followed In The Background?

- **Step 2: Sematic chck**

Once the query passes the **Syntax check**, next step is **Semantic check**.
Here, the **references** of all objects present in the query are checked along with user privileges. The check happens against **metadata** maintained in the Oracle.
Examples: 1. Is **table name** valid or such a **table** exists?
2. Are **columns** names correct?
3. Does user have **select/update/delete/insert/create** privilege?
etc. etc. etc.

So during **Syntax check**, it doesn't check for the **validity** of table names, column names, privileges etc.

Let's say, I am running this... "select * from **epm**;"

This passes **Syntax check**, though I am writing a **wrong table name**. Instead of **"emp"**, I have

written **"epm"**. But, this query is fine **syntax-wise** and hence passes the **Syntax check.** But it fails in the next step of **Semantic check** where the **object names** are verified.

But we will not be able to **notice** whether a query failed at **Syntax check** or at **Semantic check** when we run a query from SQL*Plus or any other GUI. Because, everything is handled **in one go** from user point of view and the **errors** are sent if the Query fails at any step. Otherwise, we get the **final output**.

How Does A SQL Query Execute Inside Oracle? What Are The Steps Followed In The Background?

- **Step 3: Shared pool check**

This is an important check. I am planning to write a separte lesson on this. Let us understand briefly here.

Basically, once the query passed both **Syntax check** and **Semantic check**, it goes for third check called **'Shared pool check'**.

Just to put it briefly, **'Shared pool'** is a memory segment present inside every Oracle instance, which **retains** the recently executed **SQL** and **PLSQL** code inside its **'Library Cache'** based on **LRU** algorithm.

So, if parser finds that the query is **present** inside **'Library Cache'** or **'Shared pool'**, then it is called **'Soft Parse'**. It goes to **step 5,** executes the query and sends the output to whoever requested it.

As simple as that. But, if parser finds that such a query is **not present** inside **'Shared pool'** or **'Library Cache'**, then it is called **'Hard Parse'**. So, Oracle must carry out **step 4 first** to prepare the query and then go to step 5 finally to execute the query and then send the output to the requestor.

So, the first three steps are always carried out for every SQL Query and they are less expensive.

How Does A SQL Query Execute Inside Oracle? What Are The Steps Followed In The Background?

- **Step 4:**

The step 4 is **very expensive** or **costly**. Meaning, it consumes lot of **resources** to complete this, like **CPU, I/O reads, physical reads, extra algorithms** to generate multiple queries and select the best one etc. Hence, **'Hard Parse'** degrades **performance**. We should keep this in mind always **before writing** any query. To put it

simply for now, avoid **hard coding** and use **bind variables** to take the advantage of **'Soft Parse'**. **Step 4** happens only in case of **'Hard parse'**, which is decided in **step 3** as we discussed above. We can split this into **two sub-steps**.

a. **Optimization**: The Optimizer generates multiple plans using various combinations of **joins**(Nested loop/Sort Merge/Hash), **access paths** (full table scan/index range scan/index skip scan etc) and **join orders.** Finally selects the **best plan** based on cost. **The lower the cost, the better the performance.** b. **Row source generation**: Query execution plan is generated for the **best query**

selected in the above step. This is what we see in **Explain plan**.

We can further make modifications to the query, force the Optimizer to select an **even lower cost** query by changing **access paths, joins, join orders** or using **Hints**. And we can verify if the **execution plan** got changed by again looking at its **Explain plan**.

This is called **'Performance tuning'** or **'Query Tuning'.**

• None of this happens in case of **'Soft Parse'** and hence improves **performance**.

How Does A SQL Query Execute Inside Oracle? What Are The Steps Followed In The Background?

• Step 5: Query Execution

Step 5 is again **common** for each query irrespective of whether it is **'Soft Parse'** or **'Hard Parse'.**

As we already discussed, it executes the **SQL Query** and sends the output to the requested **program** or **user**.

- To wrap up, there are **two** ways of execution or parsing namely- **'Hard parse'** and **'Soft Parse'**. And there are **5 steps** totally. **Steps 1 to 3** are common for every query or for each type of execution/parsing. **Step 4** happens only for **'Hard Parse'** way of execution and not for **'Soft Parse'** way. **Step 5** is the final and **common** step, which finally executes the **SQL Query**. Use **Explain plan** to check the **execution plan** selected by **Optimizer** and tune the query. Use **Bind variables** to enable **'Soft Parse'**.

Oracle Database Management Tools

- Login:

  $sqlplus <username>/<password> as

  sysdba $sqlplus / as sysdba
- To view the current user

  $show user
- Switching to another database user

  sql>connect <username>/<password>

  sql>connect / as sysdba

Oracle Database Management Tools

- Viewing list of tables that belongs to the current
  user sql>select <table_name> from tabs;
- To view the structure of a table
  sql>desc <table_name>;
  sql>desc employees;
- To view the last sql command executed
  sql>l;
- To re-executed the last sql statement
  sql>/;

Select Statement

- Retrieving selected data from a table sql> select table_name from tabs;

  sql> select first_name, job_id, salary from employees;
- Retrieving all data from a table

  sql> select * from employees
- Using Arithmetic Operator

  sql> select first_name, salary, 12*salary from employees;

Select Statement

- Using Concatenation Operator('|')

```
sql> select frist_name | | last_name,
        salary from employees;
```

- **Sorting and Ordered**
- Using WHERE clause

```
sql> select first_name, salary,
   hire_date from employees
    where first_name='Neena';
```

Select Statement

- Using In(value1, value2,etc)

```
sql> select first_name, salary,
        employee_id from employees
        where employee_id in(2,5,8,21,101)
```

- Using like(value that matches the given pattern) select first_name, salary, employee_id from emoloyees where first_name like ‘%am’;

Select Statement

- Using Comparision Operators(<,>,<=,>=,=,!=) select frist_name, salary, hire_date from employees where salary >=1000;
- Using Date function select sysdate from dual;

Select Statement

• Using Order by

```
sql> select frist_name, salary, hire_date
        from employees
        where salary>10000
        order by salary;
```

-- here order by clause is used for Sorting. • Using Between (value1 and value2)

```
sql> select first_name, salary, hire_date
        from employees
        where salary between 8000 and 12000;
```

Select Statement

- Wildcard character

% - matches any number of character

_ - matches any single character

```
select first_name, salary, job_id
from employees
where first_name like '_a%'
```

The value in cote may be anything like ram, rama, sam, samjana,etc. here first character should be single character(_) and follow ‘a’ the there can be any number of character(%).

Select Statement

- Using Arithmetic Operators

```
select first_name, last_name, salary, salary*12,
 hire_date from employees;
```

- Using Alias Name

```
select first_name, last_name, salary, salary*12
‘annual_salary’,  hire_date
 from employees;
```

--Here ‘annual_salary’ is the alias name given to the salary*12.

Select Statement

• Using logical operator(NOT, AND and OR is the order of precedence) -To override the order of precedence use bracket()

• OR

```
select first_name, salary, job_id
from employees
where job_id like ‘%man’
or
order by salary;
```

Select Statement

- And

```
select first_name, salary, job_id
from employees
where job_id like ‘%man’
or
job_id like ‘%rep’
and
salary > 10000
order by salary;
```

Select Statement

- NOT

```
select first_name, salary, job_id
from employees
where
job_id not like ‘%man’
or
salary > 10000
order by salary
```

Select Statement

- Using single row function

```
select first_name, job_id, salary
from employees
where lower(first_name)='ram';

select first_name, job_id, salary
from employees
where upper(first_name)='RAM';

select first_name, job_id, salary, length(first_name) 'length of first
name' from employees
```

Select Statement

• Example of single row functions are:

1. length(‘abc’)=3
2. Lower(‘Ram’)=ram
3. Upper(‘Ram’)=RAM
4. concat(‘hello’, ‘ram’)= helloram
5. Substring(‘helloram’,1,5)=hello(starts from 1st character and moves to 5th position and displays contents till there)
6. Trim(‘h’ from ‘hello’)=ello
7. Instr(‘helloworld’, ’w’)=6 (the INSTR() function returns the position of the first occurrence of a string in another string)
8. Lpad(salary,10,’*’)=********10(The LPAD() function left-pads a string with another string, to a certain length.)
9. Rpad(salary,10,*’)=125*******(The RPAD() function right-pads a string with another string, to a certain length.)

Sub Queries

```
sql>select first_name, job_id, salary
from employees
where salary > (select salary from employees
                where employee_id=12
                order by salary;)
```

• Here sub-query is executed first and the outer query is executed, • Structure of sub-query

```
Main-query
{
        Sub-Query
}
```

Aggregate functions

- Min
- Max
- Avg
- Count

- Without alias

```
sql>select min(salary), max(salary), sum(salary), count(salary),
count(employee_id) from employees;
```

- With alias

```
sql>select min(salary) ‘min-salary’, max(salary) ‘max-salary’, sum(salary) ‘total
salary’, avg(salary) ‘avg-salary’, count(employee-id) ‘total-emp’ from
employees;
```

Aggregate functions

```
select max(avg(salary)) ‘max avg sal’
from employees
group by department_id;
```

- Innermost function is evaluated first