Introduction to Information & Communication Technologies

CL-1000

Lab 05

Basic SQL

National University of Computer & Emerging Sciences – NUCES – Karachi

National University of Computer & Emerging Sciences – NUCES – Karachi

Course Code: CL-1000	Introduction to Information & Communication Technologies

Contents

1. What is SQL?

SQL (pronounced "*sequel*") is an acronym for *Structured Query Language,* a standardized language used to access and manipulate data. The history of SQL corresponds closely with the development of a **relational databases** concept published in a paper by **Dr. E. F. Codd** at **IBM in 1970**. He applied mathematical concepts to the specification of a method for data storage and access; this specification, which became the basis for relational databases, was intended to overcome the physical dependencies of the then-available database systems. The SQL language (originally called "**System R**" in the prototype and later called "**SEQUEL**") was developed by the **IBM Research Laboratory** as a **standard language** to use with relational databases. In **1979 Oracle**, then called **Relational Software**, Inc., introduced the first commercially available implementation of a relational database incorporating the SQL language.

2. Why SQL

Today, SQL is accepted as the **universal standard database access language.** Databases using the SQL language are entrusted with managing critical information affecting many aspects of our daily lives. Most applications developed today use a relational database, and **Oracle continues** to be one of the **largest** and **most popular database vendors**. Although relational databases and the SQL language are already over 45 years old, there seems to be no slowing down of the popularity of the language. Learning SQL is probably one of the best long-term investments you can make for a number of reasons:

- SQL is used by most commercial database applications.
- Although the language has evolved over the years with a large array of syntax enhancements and additions, most of the basic functionality has remained essentially unchanged.
- SQL knowledge will continue to be a fundamental skill because there is currently no
- mature and viable alternative language that accomplishes the same functionality.
- Learning Oracle's specific SQL implementation allows you great insight into the feature rich functionality of one of the largest and most successful database vendors.

3. An Introduction to Database

Before you begin to use SQL, you must know about data, databases, and **relational databases**. What is a database? A *database* is an organized collection of data. A *database management system (DBMS)* is software that allows the creation, retrieval, and manipulation of data. You use such systems to maintain patient data in a hospital, bank accounts in a bank, or inventory in a warehouse.

3.1 The Relational Database

Relational databases offer *data independence,* meaning a user does not need to know on which hard drive and file a particular piece of information is stored. The RDBMS provides users with *data consistency* and *data integrity*. For example, if an employee works in the finance department, and we know that he can work for only one department, then there should not be duplicate department records or contradicting data in the database. As you work through this lab, you will discover many of these useful and essential features.

A relational database stores data in tables, essentially a two-dimensional matrix consisting of columns and rows.

3.2 Tables

A table typically contains data about a **single subject**. Each table has a unique name that signifies the contents of the data. A database usually consists of many tables. For example, you may store data about books you read in a table called BOOK and store details about authors in the AUTHOR table.

3.3 Columns

Columns in a table organize the data further, and a table consists of at least one column. Each column represents a single, low-level detail about a particular set of data. The name of the column is unique within a table and identifies the data you find in the column. For example, the BOOK table may have a column for the title, publisher, date the book was published, and so on. The order of the columns is unimportant because SQL allows you to display data in any order you choose.

3. 4 Rows

Each row usually represents one unique set of data within a table. For example, the row in Fig with the title **"The Invisible Force"** is unique within the BOOK table. All the columns of the row represent respective data for the row. Each intersection of a column and row in a table represents a value, and some do not, as you see in the PUBLISH_DATE column. The value is said to be *null*. Null is an unknown value, so it is not even blank spaces. Nulls cannot be evaluated or compared because they are unknown.

BOOK_ID	TITLE	PUBLISHER	PUBLISH_DATE
1010	The Invisible Force	Literacy Circle	
1011	Into The Sky	Prentice Hall	10/2008
1012	Making It Possible	Life Books	08/2010

Fig: The BOOK Table

3.5 Primary Key

When working with tables, you must understand how to uniquely identify data within a table. This is the purpose of the *primary key. T*his means you find one, and only one, row in the table by looking for the primary key value. Figure shows an example of the CUSTOMER table with CUSTOMER_ID as the primary key of the table.

CUSTOMER_ID	CUSTOMER_NAME	ADDRESS	PHONE	ZIP
2010	Movers, Inc.	123 Park Lane	212-555-1212	10095
2011	Acme Mfg, Ltd.	555 Broadway	212-566-1212	10004
2012	ALR Inc.	50 Fifth Avenue	212-999-1212	10010

Figure: Primary Key Example

At first glance, you might think that the CUSTOMER_NAME column can serve as the primary key of the CUSTOMER table because it is unique. However, it is entirely possible to have customers with the same name. Therefore, the CUSTOMER_NAME column is not a good choice for the primary key. Sometimes the unique key is a system-generated sequence number; this type of key is called a *synthetic,* or *surrogate, key.* The advantage of using a surrogate key is that it is unique and does not have any inherent meaning or purpose; therefore, it is not subject to changes. In this example, the CUSTOMER_ID column is such a surrogate key.

It is best to avoid any primary keys that are subject to updates as they cause unnecessary complexity. For example, the phone number of a customer is a poor example of a primary key column choice. Though it may possibly be unique within a table, phone numbers can change and then cause a number of problems with updates of other columns that reference this column.

A table may have only one primary key, which consists of one or more columns. If the primary key contains multiple columns, it is referred to as a *composite primary key,* or *concatenated primary key*. (Choosing appropriate keys is discussed further in Chapter 12, "Create, Alter, and Drop Tables.") Oracle does not require that every table have a primary key, and there may be cases in which it is not appropriate to have one. However, it is strongly recommended that most tables have a primary key.

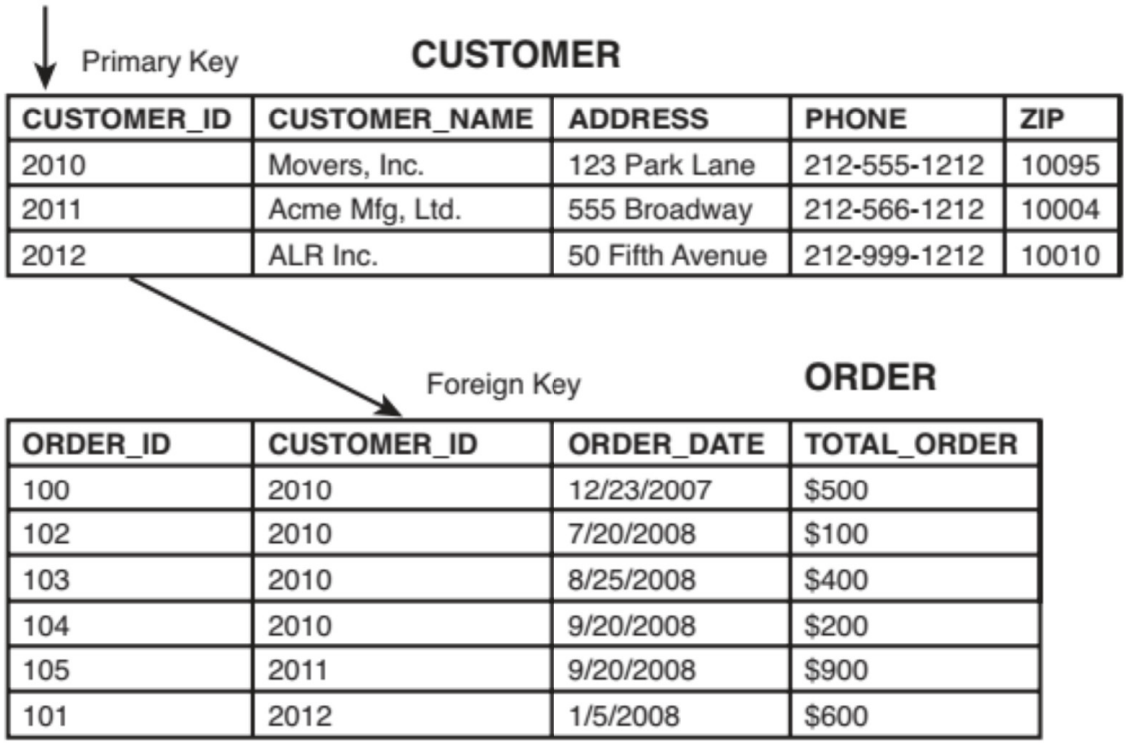
3.6 Foreign Keys

If you store customers and the customers' order information in one table, each customer's name and address is repeated for each order. The following Figure depicts such a table. Any change to the address requires the update of all the rows in the table for that individual customer.

CUSTOMER_ID	CUSTOMER_NAME	ADDRESS	PHONE	ZIP	ORDER_ID	ORDER_DATE	TOTAL_ ORDER
2010	Movers, Inc.	123 ParkLane	212-555-1212	10095	100	12/23/2007	$500
2010	Movers, Inc.	123 ParkLane	212-555-1212	10095	102	7/20/2008	$100
2010	Movers, Inc.	123 ParkLane	212-555-1212	10095	103	8/25/2008	$400
2010	Movers, Inc.	123 ParkLane	212-555-1212	10095	104	9/20/2008	$200
2011	Acme Mfg, Ltd.	555 Broadway	212-566-1212	10004	105	9/20/2008	$900
2012	ALR Inc.	50 Fifth Avenue	212-999-1212	10010	101	1/5/2008	$600

Fig: Example of CUSTOMER data mixed with ORDER data

If, however, the data is split into two tables (CUSTOMER and ORDER, as shown in Figure below) and the customer's address needs to be updated, only one row in the CUSTOMER table needs to be updated. Furthermore, separating data this way avoids the possibility of data inconsistency, whereby the data differs between the different rows.



CUSTOMER_ID	CUSTOMER_NAME	ADDRESS	PHONE	ZIP
2010	Movers, Inc.	123 Park Lane	212-555-1212	10095
2011	Acme Mfg, Ltd.	555 Broadway	212-566-1212	10004
2012	ALR Inc.	50 Fifth Avenue	212-999-1212	10010

ORDER_ID	CUSTOMER_ID	ORDER_DATE	TOTAL_ORDER
100	2010	12/23/2007	$500
102	2010	7/20/2008	$100
103	2010	8/25/2008	$400
104	2010	9/20/2008	$200
105	2011	9/20/2008	$900
101	2012	1/5/2008	$600

Fig: Primary and foreign key relationship between CUSTOMER and ORDER tables

The above illustrates how the data is split into two tables to provide data consistency. In this example, CUSTOMER_ID becomes a *foreign key* column in the ORDER table. The foreign key is the column that links the CUSTOMER and ORDER table together. You can find all orders for a particular customer by looking for the particular CUSTOMER_ID in the ORDER table. The CUSTOMER_ID corresponds to a single row in the CUSTOMER table that provides the customer-specific information.

The foreign key column CUSTOMER_ID happens to have the same column name in the ORDER table. This makes it easier to recognize the fact that the tables share common column values. Often the foreign key column and the primary key have identical column names, but it is not required.

4. Overview of SQL Language Commands

You work with tables, rows, and columns using the SQL language. SQL allows you to query data, create new data, modify existing data, and delete data. Within the SQL language you can differentiate between individual sublanguages, which are a collection of individual commands.

For example, *Data Manipulation Language* (DML) commands allow you to query, insert, update, and delete data. SQL allows you to create new database structures such as tables or modify existing ones; this subcategory of SQL language commands is called the *Data Definition Language* (DDL). Using the SQL language, you can control access to the data using *Data Control Language* (DCL) commands. The following table shows you different language categories, with examples of their respective SQL commands

Description	SQL Commands
Data Manipulation	**SELECT, UPDATE, DELETE, MERGE,**
Data Definition	**CREATE, ALTER, DROP, TRUNCATE, RENAME**
Data Control	**GRANT, REVOKE**
Transaction Control	**COMMIT, ROLLBACK, SAVEPOINT**

5. Commonly Used Datatypes

Every column in a Table has a data type, which determines what type of data can be stored. You need to know about the data types in order to use some of the comparison operators.

5.1 DATE

The **DATE** data type stores date and time information. Depending on your setup, the default display format for a date may be **DD-MON-YY**. For example, **July 4, 2009**, displays as **04-JUL-09**. There are a number of functions you can use to change the display format or to show the time. You also have menu options in SQL Developer for customizing the display.

5.2 NUMBER

Columns with the data type **NUMBER** allow only numeric data; no text, hyphens, or dashes are permitted. A column defined as **NUMBER(5,2)** can have a maximum of three digits before the decimal point and two digits after the decimal point. The first digit (**5**) is called the *precision;* the second digit (**2**) is referred to as the *scale.* The smallest allowed number is – **999.99**, and the largest is **999.99**. A column definition with a zero scale, such as **NUMBER(5)** or **NUMBER(5,0)**, allows integers in the range from **– 99,999 to 99,999**.

5.3 VARCHAR2 and CHAR

The **VARCHAR2** and **CHAR** data types store alphanumeric data (for example, text, numbers, special characters). **VARCHAR2** is the variable-length data type and the most commonly used alphanumeric data type; its maximum size is **4,000** characters. The main difference between **VARCHAR2** and **CHAR** is that the **CHAR** data type is a fixed-length data type, and any unused room is blank padded with spaces.

For example, a column defined as **CHAR(10)** and containing the four-character-length value **JOHN** in a row will have six blank characters padded at the end to make the total length **10** spaces. (If the column is stored in a **VARCHAR2(10)** column instead, it stores four characters only.) A **CHAR** column can store up to **2,000** characters.

If you want to store data containing more than **4,000** characters, you need to consider the **CLOB** data type, which allows you to store large amounts of textual data. It replaces the formerly used **LONG** data type, which is supported only for backward compatibility.

6. SQL Dialects

A **SQL dialect** refers to the specific variation or "flavor" of **Structured Query Language (SQL)** used by different relational database management systems (RDBMS). While SQL is a standard language for managing and querying databases, each RDBMS (such as MySQL, Oracle, PostgreSQL, SQL Server, etc.) may implement SQL slightly differently, leading to different **SQL dialects**.

6.1 Key Points About SQL Dialects:

1. **Core SQL Commands**: Most SQL dialects support common SQL commands like `SELECT`, `INSERT`, `UPDATE`, and `DELETE`, but they may differ in syntax or additional features.
2. **Vendor-Specific Functions and Features**: Each RDBMS may have its own functions or features. For example:
    - **MySQL** uses `LIMIT` to limit the number of rows in a query result.
    - **Oracle** uses `ROWNUM` or `FETCH FIRST` for similar functionality.
3. **Data Types**: Different dialects might have unique data types. For instance:
    - **MySQL** has data types like `TINYINT`, `DATETIME`.
    - **Oracle** uses data types like `NUMBER`, `DATE`.
4. **Performance Optimization**: SQL dialects often include vendor-specific commands and techniques for optimizing queries, indexing, and handling large datasets.

6.2 Examples of SQL Dialects:

- **MySQL Dialect**: Uses `LIMIT` for pagination, has specific `ENGINE` options like InnoDB, and functions like `NOW()` for date/time.
- **Oracle SQL**: Has its own syntax for date handling (`SYSDATE`), uses `PL/SQL` for procedural tasks, and offers advanced features like packages and sequences.
- **SQL Server Dialect**: Known as **T-SQL**, it supports TOP for limiting rows and unique transaction management functions.

Though all these dialects follow the SQL standard to some extent, their unique syntax and capabilities reflect the differences among RDBMS platforms.

7. Core SQL Statements and Clauses

7.1 SELECT Statement

The `SELECT` statement is used to query data from a table or multiple tables. It retrieves records that match specified criteria.

Syntax:

```
SELECT column1, column2
FROM table_name;
```

Example:

```
SELECT first_name, last_name
FROM employees;
```

7.2 WHERE Clause

The `WHERE` clause is used to filter records. It specifies conditions that the records must meet to be included in the result set.

Syntax:

```
SELECT column1, column2
FROM table_name
WHERE condition;
```

Example:

```
SELECT first_name, last_name
FROM employees
WHERE department_id = 90;
```

7.3 ORDER BY Clause

`ORDER BY` is used to sort the result set in ascending (`ASC`) or descending (`DESC`) order based on one or more columns.

Syntax:

```
SELECT column1, column2
FROM table_name
ORDER BY column1 [ASC|DESC];
```

Example:

```
SELECT first_name, last_name
FROM employees
ORDER BY last_name ASC;
```

7.4 INSERT INTO Statement

The `INSERT INTO` statement is used to add new rows to a table.

Syntax:

```
INSERT INTO table_name (column1, column2)
VALUES (value1, value2);
```

Example:

```
INSERT INTO departments (department_id, department_name)
VALUES (280, 'Marketing');
```

7.5 UPDATE Statement

`UPDATE` is used to modify existing records in a table based on certain conditions.

Syntax:

```
UPDATE table_name
SET column1 = value1
WHERE condition;
```

Example:

```
UPDATE employees
SET salary = 6000
WHERE employee_id = 101;
```

7.6 DELETE Statement

The `DELETE` statement is used to remove existing records from a table based on a condition.

Syntax:

```
DELETE FROM table_name
WHERE condition;
```

Example:

```
DELETE FROM employees
WHERE employee_id = 102;
```

7.7 JOIN Clause

Joins combine rows from two or more tables based on a related column between them.

Types:

- `INNER JOIN`: Returns only matching rows between tables.
- `LEFT JOIN`: Returns all rows from the left table, and matching rows from the right.
- `RIGHT JOIN`: Returns all rows from the right table, and matching rows from the left.

Example:

```
SELECT e.first_name, e.last_name, d.department_name
FROM employees e
INNER JOIN departments d
    ON e.department_id = d.department_id;
```

||' '|| is used to join two strings as SELECT first_name||' '|| last_name AS full_name;

Here, we join the employees and departments tables to display employee names and their corresponding department names.

7.8 GROUP BY Clause

The `GROUP BY` clause is used with aggregate functions (like `COUNT`, `SUM`, `AVG`) to group result-set rows based on one or more columns.

Syntax:

```
SELECT column1, COUNT(column2)
FROM table_name
GROUP BY column1;
```

Example:

```
SELECT department_id, COUNT(employee_id)
FROM employees
GROUP BY department_id;
```

7.9 HAVING Clause

`HAVING` is used to filter groups formed by the `GROUP BY` clause, similar to how `WHERE` filters rows.

Syntax:

```
SELECT column1, aggregate_function(column2)
FROM table_name
GROUP BY column1
HAVING aggregate_function(column2) condition;
```

Example:

```sql
SELECT department_id, COUNT(employee_id)
FROM employees
GROUP BY department_id
HAVING COUNT(employee_id) > 5;
```

MIN AND MAX FUNCTION IS USED TO GIVE MINIMUM AND MAXIMUM VALUE FROM TABLE

Here, we are retrieving the departments, that have more than 5 employees.

7.10 LIMIT Clause

The `LIMIT` clause is used to specify the number of records to return from a query.

Syntax:

```sql
SELECT column1, column2
FROM table_name
LIMIT number_of_rows;
```

Example:

```sql
SELECT first_name, last_name
FROM employees
LIMIT 10;
```

7.11 DISTINCT Clause

`DISTINCT` is used to remove duplicate records from the result set.

Syntax:

```sql
SELECT DISTINCT column1
FROM table_name;
```

Example:

```sql
SELECT DISTINCT department_id
FROM employees;
```