**SQL CONSTRAINTS**

**Constraints are the rules enforced on data columns on table**.

**These are used to limit the type of data that can go into a table**.

**This ensures the accuracy and reliability** of the data in the database.

Constraints could **be column level or table level**. Column level constraints are applied only to one column, whereas table level constraints are applied to the whole table.

Following are commonly used constraints available in SQL.

* NOT NULL Constraint: Ensures that a column cannot have NULL value.
* DEFAULT Constraint: Provides a default value for a column when none is specified.
* UNIQUE Constraint: Ensures that all values in a column are different.
* PRIMARY Key: Uniquely identified each rows/records in a database table.
* FOREIGN Key: Uniquely identified a rows/records in any another database table.
* CHECK Constraint: The CHECK constraint ensures that all values in a column satisfy certain conditions.
* INDEX: Use to create and retrieve data from the database very quickly.

Constraints can be specified when a table is created with the CREATE TABLE statement or you can use ALTER TABLE statement to create constraints even after the table is created.

**Dropping Constraints:**

Any constraint that you have defined can be dropped using the ALTER TABLE command with the DROP CONSTRAINT option.

For example, to drop the primary key constraint in the EMPLOYEES table, you can use the following command:

ALTER TABLE EMPLOYEES DROP CONSTRAINT EMPLOYEES\_PK;

Some implementations may provide shortcuts for dropping certain constraints. For example, to drop the primary key constraint for a table in Oracle, you can use the following command:

ALTER TABLE EMPLOYEES DROP PRIMARY KEY;

Some implementations allow you to disable constraints. Instead of permanently dropping a constraint from the database, you may want to temporarily disable the constraint and then enable it later.

**Integrity Constraints:**

Integrity constraints are used to ensure accuracy and consistency of data in a relational database. Data integrity is handled in a relational database through the concept of referential integrity.

There are many types of integrity constraints that play a role in referential integrity (RI). These constraints include Primary Key, Foreign Key, Unique Constraints and other constraints mentioned above.

**SQL JOINS**

The SQL **Joins** clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

Consider the following two tables, (a) CUSTOMERS table is as follows:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

(b) Another table is ORDERS as follows:

+-----+---------------------+-------------+--------+

|OID | DATE | CUSTOMER\_ID | AMOUNT |

+-----+---------------------+-------------+--------+

| 102 | 2009-10-08 00:00:00 | 3 | 3000 |

| 100 | 2009-10-08 00:00:00 | 3 | 1500 |

| 101 | 2009-11-20 00:00:00 | 2 | 1560 |

| 103 | 2008-05-20 00:00:00 | 4 | 2060 |

+-----+---------------------+-------------+--------+

Now, let us join these two tables in our SELECT statement as follows:

SQL> SELECT ID, NAME, AGE, AMOUNT

FROM CUSTOMERS, ORDERS

WHERE CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;

This would produce the following result:

+----+----------+-----+--------+

| ID | NAME | AGE | AMOUNT |

+----+----------+-----+--------+

| 3 | kaushik | 23 | 3000 |

| 3 | kaushik | 23 | 1500 |

| 2 | Khilan | 25 | 1560 |

| 4 | Chaitali | 25 | 2060 |

+----+----------+-----+--------+

Here, it is noticeable that the join is performed in the WHERE clause. Several operators can be used to join tables, such as =, <, >, <>, <=, >=, !=, BETWEEN, LIKE, and NOT; they can all be used to join tables. However, the most common operator is the equal symbol.

## SQL Join Types:

There are different types of joins available in SQL:

* INNER JOIN: returns rows when there is a match in both tables.
* LEFT JOIN: returns all rows from the left table, even if there are no matches in the right table.
* RIGHT JOIN: returns all rows from the right table, even if there are no matches in the left table.
* FULL JOIN: returns rows when there is a match in one of the tables.
* SELF JOIN: is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.
* CARTESIAN JOIN: returns the Cartesian product of the sets of records from the two or more joined tables.

**UNION CLAUSE**

The SQL **UNION** clause/operator is used to combine the results of two or more SELECT statements without returning any duplicate rows.

To use UNION, each SELECT must have the same number of columns selected, the same number of column expressions, the same data type, and have them in the same order, but they do not have to be the same length.

## Syntax:

The basic syntax of **UNION** is as follows:



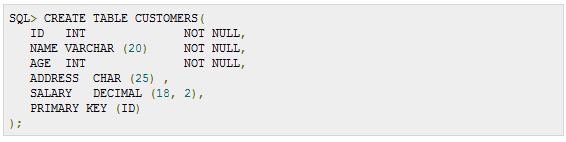
**SQL NULL VALUES**

The SQL **NULL** is the term used to represent a missing value. A NULL value in a table is a value in a field that appears to be blank.

A field with a NULL value is a field with no value. It is very important to understand that a NULL value is different than a zero value or a field that contains spaces.

**Syntax:**

The basic syntax of **NULL** while creating a table:



Here, **NOT NULL** signifies that column should always accept an explicit value of the given data type. There are two columns where we did not use NOT NULL, which means these columns could be NULL.

A field with a NULL value is one that has been left blank during record creation.

**SQL ALIAS SYNTAX**

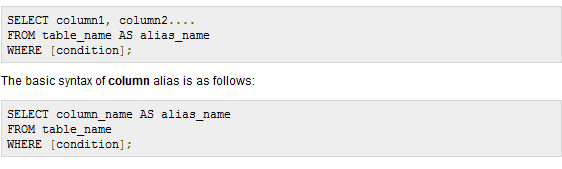
You can rename a table or a column temporarily by giving another name known as alias.

The use of table aliases means to rename a table in a particular SQL statement. The renaming is a temporary change and the actual table name does not change in the database.

The column aliases are used to rename a table's columns for the purpose of a particular SQL query.

**Syntax:**

The basic syntax of **table** alias is as follows:



**SQL INDEXES**

Indexes are special lookup tables that the database search engine can use to speed up data retrieval. Simply put, an index is a pointer to data in a table. An index in a database is very similar to an index in the back of a book.

For example, if you want to reference all pages in a book that discuss a certain topic, you first refer to the index, which lists all topics alphabetically and are then referred to one or more specific page numbers.

An index helps speed up SELECT queries and WHERE clauses, but it slows down data input, with UPDATE and INSERT statements. Indexes can be created or dropped with no effect on the data.

Creating an index involves the CREATE INDEX statement, which allows you to name the index, to specify the table and which column or columns to index, and to indicate whether the index is in ascending or descending order.

Indexes can also be unique, similar to the UNIQUE constraint, in that the index prevents duplicate entries in the column or combination of columns on which there's an index.

**The CREATE INDEX Command:**

The basic syntax of **CREATE INDEX** is as follows:

CREATE INDEX index\_name ON table\_name;

## Single-Column Indexes:

A single-column index is one that is created based on only one table column. The basic syntax is as follows:

CREATE INDEX index\_name

ON table\_name (column\_name);

## Unique Indexes:

Unique indexes are used not only for performance, but also for data integrity. A unique index does not allow any duplicate values to be inserted into the table. The basic syntax is as follows:

CREATE UNIQUE INDEX index\_name

on table\_name (column\_name);

## Composite Indexes:

A composite index is an index on two or more columns of a table. The basic syntax is as follows:

CREATE INDEX index\_name

on table\_name (column1, column2);

**SQL ALTER TABLE COMMAND**

The SQL **ALTER TABLE** command is used to add, delete or modify columns in an existing table.

You would also use ALTER TABLE command to add and drop various constraints on a an existing table.

## Syntax:

The basic syntax of **ALTER TABLE** to add a new column in an existing table is as follows:

ALTER TABLE table\_name ADD column\_name datatype;

The basic syntax of ALTER TABLE to **DROP COLUMN** in an existing table is as follows:

ALTER TABLE table\_name DROP COLUMN column\_name;

The basic syntax of ALTER TABLE to change the **DATA TYPE** of a column in a table is as follows:

ALTER TABLE table\_name MODIFY COLUMN column\_name datatype;

The basic syntax of ALTER TABLE to add a **NOT NULL** constraint to a column in a table is as follows:

ALTER TABLE table\_name MODIFY column\_name datatype NOT NULL;

The basic syntax of ALTER TABLE to **ADD UNIQUE CONSTRAINT** to a table is as follows:

ALTER TABLE table\_name

ADD CONSTRAINT MyUniqueConstraint UNIQUE(column1, column2...);

The basic syntax of ALTER TABLE to **ADD CHECK CONSTRAINT** to a table is as follows:

ALTER TABLE table\_name

ADD CONSTRAINT MyUniqueConstraint CHECK (CONDITION);

The basic syntax of ALTER TABLE to **ADD PRIMARY KEY** constraint to a table is as follows:

ALTER TABLE table\_name

ADD CONSTRAINT MyPrimaryKey PRIMARY KEY (column1, column2...);

The basic syntax of ALTER TABLE to **DROP CONSTRAINT** from a table is as follows:

ALTER TABLE table\_name

DROP CONSTRAINT MyUniqueConstraint;

If you're using MySQL, the code is as follows:

ALTER TABLE table\_name

DROP INDEX MyUniqueConstraint;

The basic syntax of ALTER TABLE to **DROP PRIMARY KEY** constraint from a table is as follows:

ALTER TABLE table\_name

DROP CONSTRAINT MyPrimaryKey;

If you're using MySQL, the code is as follows:

ALTER TABLE table\_name

DROP PRIMARY KEY;

**SQL TRUNCATE TABLE COMMAND**

The SQL **TRUNCATE TABLE** command is used to delete complete data from an existing table.

You can also use DROP TABLE command to delete complete table but it would remove complete table structure form the database and you would need to re-create this table once again if you wish you store some data.

## Syntax:

The basic syntax of **TRUNCATE TABLE** is as follows:

TRUNCATE TABLE table\_name;

**SQL HAVING CLAUSE**

The HAVING clause enables you to specify conditions that filter which group results appear in the final results.

The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

## Syntax:

The following is the position of the HAVING clause in a query:

SELECT

FROM

WHERE

GROUP BY

HAVING

ORDER BY

The HAVING clause must follow the GROUP BY clause in a query and must also precede the ORDER BY clause if used. The following is the syntax of the SELECT statement, including the HAVING clause:

SELECT column1, column2

FROM table1, table2

WHERE [ conditions ]

GROUP BY column1, column2

HAVING [ conditions ]

ORDER BY column1, column2

**SQL TRANSACTIONS**

A transaction is a unit of work that is performed against a database. Transactions are units or sequences of work accomplished in a logical order, whether in a manual fashion by a user or automatically by some sort of a database program.

A transaction is the propagation of one or more changes to the database. For example, if you are creating a record or updating a record or deleting a record from the table, then you are performing transaction on the table. It is important to control transactions to ensure data integrity and to handle database errors.

Practically, you will club many SQL queries into a group and you will execute all of them together as a part of a transaction.

## The COMMIT Command:

The COMMIT command is the transactional command used to save changes invoked by a transaction to the database.

The COMMIT command saves all transactions to the database since the last COMMIT or ROLLBACK command.

The syntax for COMMIT command is as follows:

COMMIT;

## The ROLLBACK Command:

The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database.

The ROLLBACK command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.

The syntax for ROLLBACK command is as follows:

ROLLBACK;

**SUBQUERIES**

A Subquery or Inner query or Nested query is a query within another SQL query and embedded within the WHERE clause.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN etc.

There are a few rules that subqueries must follow:

* Subqueries must be enclosed within parentheses.
* A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
* An ORDER BY cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY can be used to perform the same function as the ORDER BY in a subquery.
* Subqueries that return more than one row can only be used with multiple value operators, such as the IN operator.
* The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
* A subquery cannot be immediately enclosed in a set function.
* The BETWEEN operator cannot be used with a subquery; however, the BETWEEN operator can be used within the subquery.

## Subqueries with the SELECT Statement:

Subqueries are most frequently used with the SELECT statement. The basic syntax is as follows:

SELECT column\_name [, column\_name ]

FROM table1 [, table2 ]

WHERE column\_name OPERATOR

(SELECT column\_name [, column\_name ]

FROM table1 [, table2 ]

[WHERE])

**HANDLING DUPLICATES**

There may be a situation when you have multiple duplicate records in a table. While fetching such records, it makes more sense to fetch only unique records instead of fetching duplicate records.

The SQL **DISTINCT** keyword, which we already have discussed, is used in conjunction with SELECT statement to eliminate all the duplicate records and fetching only unique records.

## Syntax:

The basic syntax of DISTINCT keyword to eliminate duplicate records is as follows:

SELECT DISTINCT column1, column2,.....columnN

FROM table\_name

WHERE [condition]