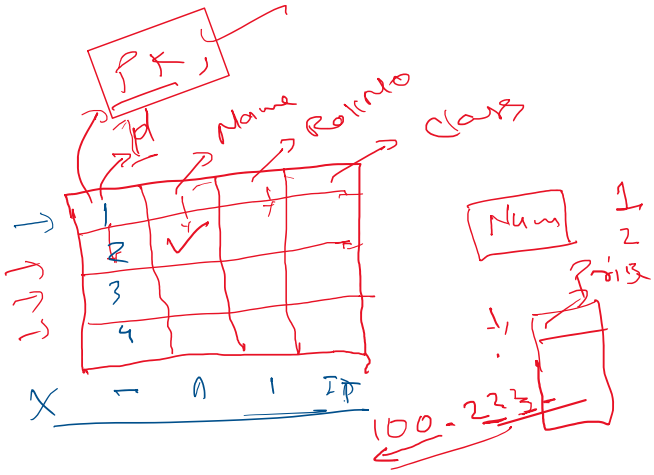
Session- 20 Database

22 August 2024 10:22 AM

Agenda

- **Keys, Constraints and Clauses**
- **Operators and AggregateFunctions**
- **Basic SQL Queries**
- **Data Filtering and Sorting Data**
- **Joining Tables**
- **Joins -Inner Join -Left Join -Right Join -Full Outer Join -Self**
- **Join Subqueries Nested Queries**



SQL Constraints

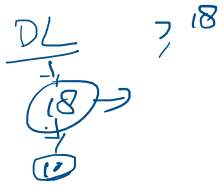
SQL constraints are used to specify rules for the data in a table.

Constraints 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 table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

- `NOT NULL` - Ensures that a column cannot have a NULL value
- `UNIQUE` - Ensures that all values in a column are different
- `PRIMARY KEY` - A combination of a `NOT NULL` and `UNIQUE`. Uniquely identifies each row in a table
- `FOREIGN KEY` - Prevents actions that would destroy links between tables
- `CHECK` - Ensures that the values in a column satisfies a specific condition
- `DEFAULT` - Sets a default value for a column if no value is specified
- `CREATE INDEX` - Used to create and retrieve data from the database very quickly



SQL NOT NULL Constraint

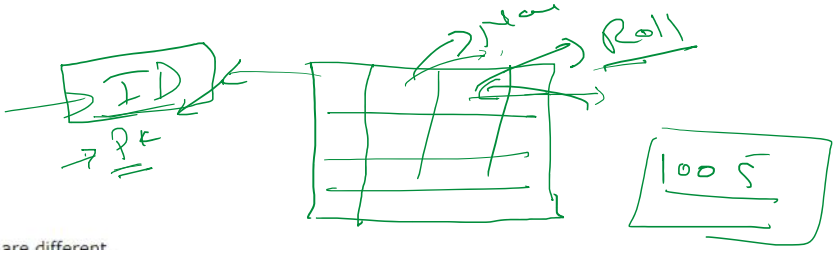
By default, a column can hold NULL values.

The `NOT NULL` constraint enforces a column to NOT accept NULL values.

This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255) NOT NULL,
    Age int
);
```

```
ALTER TABLE Persons
MODIFY COLUMN Age int NOT NULL;
```



SQL UNIQUE Constraint

The UNIQUE constraint ensures that all values in a column are different

SQL UNIQUE Constraint

The UNIQUE constraint ensures that all values in a column are different.

Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint.

However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    UNIQUE (ID)
);
```

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT UC_Person UNIQUE (ID,LastName)
);
```

```
ALTER TABLE Persons
ADD CONSTRAINT UC_Person UNIQUE (ID,LastName);
```

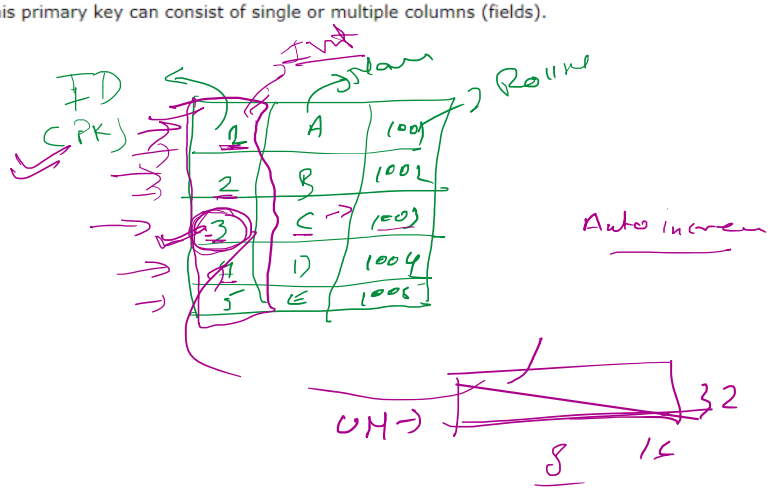
SQL PRIMARY KEY Constraint

The PRIMARY KEY constraint uniquely identifies each record in a table.

Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```



```
ALTER TABLE Persons
ADD PRIMARY KEY (ID);
```

SQL FOREIGN KEY Constraint

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

SQL FOREIGN KEY Constraint

The `FOREIGN KEY` constraint is used to prevent actions that would destroy links between tables.

A `FOREIGN KEY` is a field (or collection of fields) in one table, that refers to the `PRIMARY KEY` in another table.

The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

Look at the following two tables:

Persons Table

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

Orders Table

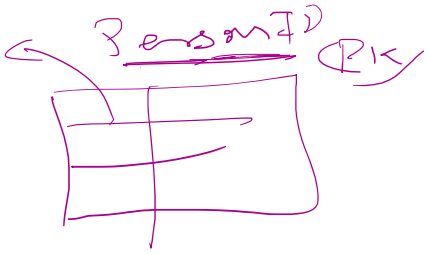
OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.

The "PersonID" column in the "Persons" table is the `PRIMARY KEY` in the "Persons" table.

The "PersonID" column in the "Orders" table is a `FOREIGN KEY` in the "Orders" table.

The `FOREIGN KEY` constraint prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the parent table.



```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

```
ALTER TABLE Orders
ADD CONSTRAINT FK_PersonOrder
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

SQL CHECK Constraint

SQL CHECK Constraint

The `CHECK` constraint is used to limit the value range that can be placed in a column.

If you define a `CHECK` constraint on a column it will allow only certain values for this column.

If you define a `CHECK` constraint on a table it can limit the values in certain columns based on values in other columns in the row.

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CHECK (Age>=18)
);
```

```
ALTER TABLE Persons
ADD CHECK (Age>=18);
```

SQL DEFAULT Constraint

The `DEFAULT` constraint is used to set a default value for a column.

The default value will be added to all new records, if no other value is specified.

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'Sandnes'
);
```

```
ALTER TABLE Persons
ALTER City SET DEFAULT 'Sandnes';
```

select column from Table where

Introduction to SQL Clauses

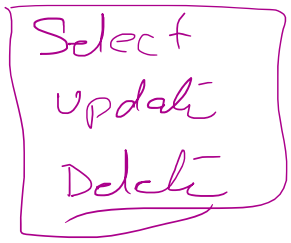
MySQL queries are SQL functions that help us to access a particular set of records from a database table. We can request any information or data from the database using the clauses or, let's say, SQL statements. For example, SQL Clauses receives a conditional expression that can be a column name or valid term involving columns where this supports the MySQL functions to calculate the result values for a table in the database.

There are generally five kinds of SQL Clauses in MySQL Server. They are listed as follows:

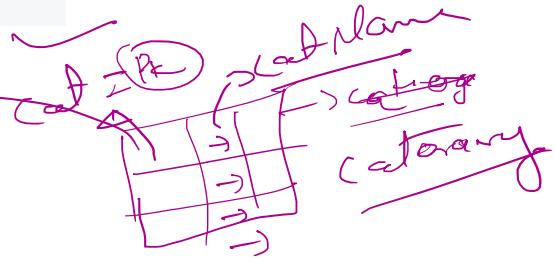
- WHERE Clause
- ORDER BY clause
- HAVING Clause
- TOP Clause
- GROUP BY Clause

1. SQL WHERE Clause

In MySQL, we use the SQL SELECT statement to select data from a table in the database. Here, the WHERE clause allows filtering certain records that exactly match a specified condition. Thus, it helps us to fetch only the necessary data from the database that satisfies the given expressional conditions. The WHERE clause is used with SELECT statement as well as with UPDATE, DELETE type statements and aggregate functions to restrict the no. of records to be retrieved by the table. We can also use logical or comparison operators such as LIKE, <, >, =, etc. with WHERE clause to fulfill certain conditions.




```
SELECT Column1,….ColumnN From Table_name WHERE [condition];
```



BOOKID	BOOKNAME	PRICE	NUMPAGE	CATID	LANG
1	3D Graphics	200	80	1	English
2	Conceptual Physics	500	200	3	English
3	Health Nursing	500	200	2	Hindi
4	Networking	300	100	2	English
5	Human Anatomy	100	50	2	Hindi

Query:

```
SELECT BookName, Price, Lang From Books WHERE CatID >1;
```

Output:

BOOKNAME	PRICE	LANG
Health Nursing	500	Hindi
Networking	300	English
Human Anatomy	100	Hindi
Conceptual Physics	500	English

Download CSV
4 rows selected.

Query:

```
SELECT Price, NumPage From Books WHERE BookName='Networking';
```

```
SELECT Price, NumPage From Books WHERE BookName='Networking';
```

Output:

PRICE	NUMPAGE
300	100

Download CSV

2. SQL ORDER BY Clause

The ORDER BY clause is used in SQL for sorting records. It is used to arrange the result set either in ascending or descending order. When we query using SELECT statement the result is not in an ordered form. Hence, the result rows can be sorted when we combine the SELECT statement with the ORDER BY clause.

```
SELECT column1, …,columnN FROM TableName ORDER BY column1,...,column ASC|DESC;
```

Query:

```
SELECT BookName, Price From Books ORDER BY Price ASC;
```

Output:

BOOKNAME	PRICE
Human Anatomy	100
3D Graphics	200
Networking	300
Health Nursing	500
Conceptual Physics	500

Download CSV
5 rows selected.

```
SELECT BookName, NumPage From Books ORDER BY NumPage DESC;
```

Output:

BOOKNAME	NUMPAGE
Conceptual Physics	200
Health Nursing	200
Networking	100
3D Graphics	80
Human Anatomy	50

Download CSV
5 rows selected.

3. SQL GROUP BY Clause

The GROUP BY clause is used to group rows that have the same values in the result set. Like if we find the names of books from the table grouped by CatID.

Query:

```
SELECT Column FROM Table WHERE condition GROUP BY Column [ORDER BY Column];
```

This clause is generally used with aggregate functions that allow grouping the query result rows by multiple columns. The aggregate functions are COUNT, MAX, MIN, SUM, AVG, etc.

```
SELECT COUNT(BookName), CatID From Books GROUP BY CatID;
```

Output:

COUNT(BOOKNAME)	CATID
1	1
3	2
1	3

Download CSV
3 rows selected.

The SQL GROUP BY clause returns the aggregated value applying the functions on the columns of the table. The above screenshot shows that the result is returned grouped by CatID where no. of BookName present in those CatID is fetched.

4. SQL HAVING Clause

Actually, this clause is introduced to apply functions in the query with the WHERE clause. In SQL, the HAVING clause was added because the WHERE clause could not be applied with aggregate functions.

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

```
SELECT COUNT (CatID), Lang From Books GROUP BY Lang HAVING COUNT(CATID) <3;
```

Output:

COUNT(CATID)	LANG
2	Hindi

Download CSV

Link for more info https://www.w3schools.com/SQL/sql_having.asp

5. SQL TOP Clause

The TOP clause is used to determine the number of record rows to be shown in the result. This TOP clause is used with SELECT statement specially implemented on large tables with many records. But the clause is not supported in many database systems, like MySQL supports the LIMIT clause to select limited no. of rows and in Oracle ROWNUM is used.

For SQL Server / MS Access Query:

```
SELECT TOP no|percentage ColumnName(s) FROM TableName WHERE condition;
```

MySQL Query:

```
SELECT ColumnName(s) FROM TableName WHERE condition LIMIT no;
```

Oracle Query:

```
SELECT ColumnName(s) FROM TableName WHERE ROWNUM <= no;
```

```
SELECT TOP 3 * FROM Books;
SELECT * FROM Books LIMIT 3;
SELECT * FROM Books WHERE ROWNUM <= 3;
```

Output:

BOOKID	BOOKNAME	PRICE	NUMPAGE	CATID	LANG
1	3D Graphics	200	80	1	English
2	Health Nursing	500	200	2	Hindi
3	Networking	300	100	2	English

Download CSV
3 rows selected.