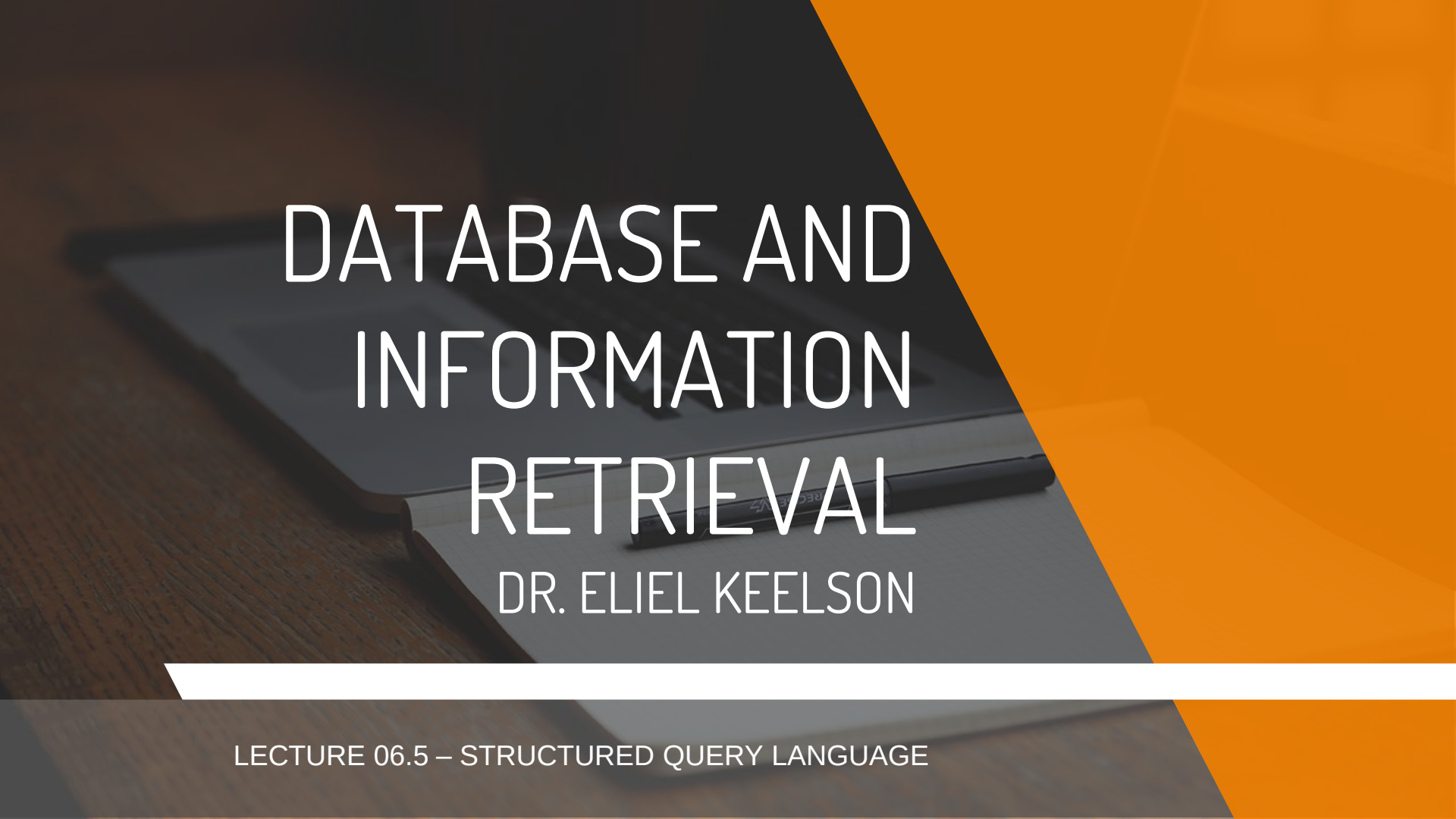

DATABASE AND INFORMATION RETRIEVAL

DR. ELIEL KEELSON

LECTURE 06.5 – STRUCTURED QUERY LANGUAGE

SQL

- These slides presents a summary of most the SQL concepts that have already been covered

- It also introduces some few relevant ones with detailed examples

SQL

- SQL stands for Structured Query Language
- SQL is a standard language for storing, manipulating and retrieving data in databases.
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

SQL

- Although SQL is an ANSI/ISO standard, there are different versions of the SQL language.

- However, to be compliant with the ANSI standard, they all support at least the major commands (such as SELECT, UPDATE, DELETE, INSERT, WHERE) in a similar manner.

- Most of the SQL-based DBMS also have their own proprietary extensions in addition to the SQL standard!

SQL

- SQL keywords are NOT case sensitive: for that matter, the keyword select is the same as SELECT

- Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.

The SQL CREATE DATABASE Statement

- The CREATE DATABASE statement is used to create a new SQL database.

- Syntax

```
CREATE DATABASE databasename;
```

- Example: Creating a database by name “testDB”

```
CREATE DATABASE testDB;
```

SQL DROP DATABASE Statement

- The DROP DATABASE statement is used to drop an existing SQL database.

- Syntax

DROP DATABASE *databasename*;

- Example: Dropping an existing database by name “testDB”

DROP DATABASE testDB;

SQL CREATE TABLE Statement

- The CREATE TABLE statement is used to create a new table in a database.

- Syntax

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

SQL CREATE TABLE Statement

- The column parameters specify the names of the columns of the table.

- The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

SQL CREATE TABLE Statement

- Example: The following example creates a table called "Persons" that contains five columns: PersonID, LastName, FirstName, Address, and City:

```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

SQL CREATE TABLE Statement

- The PersonID column is of type int and will hold an integer.

- The LastName, FirstName, Address, and City columns are of type varchar and will hold characters, and the maximum length for these fields is 255 characters.

Create Table Duplicates Using Another Table

- A copy of an existing table can be created using a combination of the CREATE TABLE statement and the SELECT statement.
- The new table gets the same column definitions.
- All columns or specific columns can be selected.
- If you create a new table using an existing table, the new table will be filled with the existing values from the old table – a copy of the existing table

Create Table Duplicates Using Another Table

- Syntax

```
CREATE TABLE new_table_name AS
    SELECT column1, column2,...
    FROM existing_table_name
    WHERE ....;
```

- Example

```
CREATE TABLE personsCopy AS
SELECT PersonID, LastName, City
FROM Persons;
```

SQL DROP TABLE Statement

- The DROP TABLE statement is used to drop/delete an existing table in a database.

- Syntax

```
DROP TABLE table_name;
```

- Example: The following SQL statement drops the existing table "Shippers":

```
DROP TABLE Shippers;
```

SQL TRUNCATE Statement

- The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

- Syntax

TRUNCATE TABLE *table_name*;

SQL ALTER TABLE Statement

- The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

- The ALTER TABLE statement can also be used to add and drop various constraints on an existing table.

SQL ALTER TABLE Statement – ADD Column

- Syntax

ALTER TABLE *table_name*
ADD *column_name datatype*;

- Example

ALTER TABLE *personsCopy*
ADD *DateofBirth date*;

SQL ALTER TABLE Statement – ALTER/MODIFY Column

- Syntax

ALTER TABLE *table_name*
MODIFY COLUMN *column_name datatype*;

- Example

ALTER TABLE *personsCopy*
MODIFY COLUMN *DateofBirth year*;

SQL ALTER TABLE Statement – DROP Column

- Syntax

ALTER TABLE *table_name*
DROP COLUMN *column_name*;

- Example

ALTER TABLE *personsCopy*
DROP COLUMN *DateofBirth*;

SQL Constraints

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

- Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.

SQL Constraints

- Syntax

```
CREATE TABLE table_name (
    column1 datatype constraint,
    column2 datatype constraint,
    column3 datatype constraint,
    ....
);
```

SQL Constraints

- 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.

SQL Constraints

- Constraints can be column level or table level.

- Column level constraints apply to a column, and table level constraints apply to the whole table.

SQL Constraints

- The following constraints are commonly used in SQL:

1. **NOT NULL** - Ensures that a column cannot have a NULL value
2. **UNIQUE** - Ensures that all values in a column are different
3. **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table

SQL Constraints

4. **FOREIGN KEY** - Uniquely identifies a row/record in another table
5. **CHECK** - Ensures that all values in a column satisfies a specific condition
6. **DEFAULT** - Sets a default value for a column when no value is specified
7. **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.

SQL NOT NULL Constraint

- The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values:

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

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.

SQL UNIQUE Constraint

- 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.

SQL UNIQUE Constraint

- The following SQL creates a UNIQUE constraint on the "ID" column when the "Persons" table is created:

```
CREATE TABLE Persons101(
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    UNIQUE (ID)
);
```

SQL UNIQUE Constraint

- Since the ID column in the Persons101 table has both the UNIQUE and NOT NULL constraints, it becomes the primary key of the table.

- You could verify by describe the table using the following command:

```
DESCRIBE  Persons101;
```

SQL UNIQUE Constraint

- To create a UNIQUE constraint on the “FirstName” column when the table is already created, use the following SQL:

```
ALTER TABLE Persons101
ADD UNIQUE (FirstName);
```

SQL UNIQUE Constraint

- To create a UNIQUE constraint on the “FirstName" column when the table is already created, use the following SQL:

```
ALTER TABLE Persons101
ADD UNIQUE (FirstName);
```

- You could verify this by describe the table using the following command:

```
DESCRIBE  Persons101;
```

SQL UNIQUE Constraint

- if you want to drop a unique constraint in MySQL, you have to drop an index rather than the unique constraint itself.
- SQL considers a unique constraint to be an index.
- To drop a UNIQUE constraint on the recently modified “FirstName” column, use the following SQL:

```
ALTER TABLE Persons101 DROP INDEX FirstName;
```

SQL PRIMARY KEY Constraint

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

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

- A table can have only one primary key, which may consist of single or multiple fields (composite PRIMARY KEY).

SQL PRIMARY KEY Constraint

- The following SQL creates a PRIMARY KEY on the "ID" column when the "Persons102" table is created:

```
CREATE TABLE Persons102 (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

SQL PRIMARY KEY Constraint

- To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

```
CREATE TABLE Persons103 (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
);
```

SQL PRIMARY KEY Constraint

- To drop a PRIMARY KEY constraint, use the following SQL:

```
ALTER TABLE Persons103
DROP PRIMARY KEY;
```

SQL PRIMARY KEY Constraint

- To create a PRIMARY KEY constraint on the "ID" column when the table is already created, use the following SQL:

```
ALTER TABLE Persons103
ADD PRIMARY KEY (ID);
```

SQL PRIMARY KEY Constraint

- To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns of an existing table, use the following SQL syntax:

```
ALTER TABLE Persons103
ADD CONSTRAINT PK_Person PRIMARY KEY (ID,LastName);
```

SQL FOREIGN KEY Constraint

- A FOREIGN KEY is a key used to link two tables together.
- A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.
- The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

SQL FOREIGN KEY Constraint

- 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

SQL FOREIGN KEY Constraint

- 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.

SQL FOREIGN KEY Constraint

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

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

SQL FOREIGN KEY Constraint

- The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:

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

SQL FOREIGN KEY Constraint

- To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

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

SQL FOREIGN KEY Constraint

- To create a FOREIGN KEY constraint on the "PersonID" column when the "Orders" table is already created, use the following SQL:

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

SQL FOREIGN KEY Constraint

- To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

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

SQL FOREIGN KEY Constraint

- To drop a FOREIGN KEY constraint, use the following SQL:

```
ALTER TABLE Orders
DROP FOREIGN KEY FK_PersonOrder;
```

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 single column it allows 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.

SQL CHECK Constraint

- The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that you can not have any person below 18 years:

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

SQL CHECK Constraint

- To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255),
    CONSTRAINT CHK_Person CHECK (Age>=18 AND City='Sandnes')
);
```

SQL CHECK Constraint

- To create a CHECK constraint on the "Age" column when the table is already created, use the following SQL:

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

SQL CHECK Constraint

- To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

```
ALTER TABLE Persons
ADD CONSTRAINT CHK_PersonAge CHECK (Age>=18 AND City='Accra');
```

SQL DEFAULT Constraint

- The DEFAULT constraint is used to provide a default value for a column.

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

SQL DEFAULT Constraint

- The following SQL sets a DEFAULT value for the "City" column when the "Persons104" table is created:

```
CREATE TABLE Persons104 (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'Accra'
);
```

SQL DEFAULT Constraint

- The DEFAULT constraint can also be used to insert system values, by using functions like NOW():

```
CREATE TABLE Orders104 (
    ID int NOT NULL,
    OrderNumber int NOT NULL,
    OrderDate datetime DEFAULT NOW()
);
```

SQL DEFAULT Constraint

- To create a DEFAULT constraint on the “Age” column when the table is already created, use the following SQL:

```
ALTER TABLE Persons104
ALTER Age SET DEFAULT 18;
```

SQL DEFAULT Constraint

- To drop a DEFAULT constraint, use the following SQL:

```
ALTER TABLE Persons104
ALTER City DROP DEFAULT;
```

SQL CREATE INDEX Statement

- The CREATE INDEX statement is used to create indexes in tables.
- Indexes are used to retrieve data from the database very fast.
- The users cannot see the indexes, they are just used to speed up searches/queries.
- Note that updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.

SQL CREATE INDEX Statement

- The following SQL syntax creates an index on a table. It allows for duplicate values:

CREATE INDEX *index_name*
ON *table_name* (*column1*, *column2*, ...);

Example:

CREATE INDEX idx_lastnameCity
ON Persons104 (LastName, City);

SQL CREATE INDEX Statement

- The following SQL syntax creates an index on a table. It does not allow for duplicate values:

```
CREATE UNIQUE INDEX index_name
ON table_name (column1, column2, ...);
```

Example:

```
CREATE UNIQUE INDEX idx_orderNumber
ON Orders104 (OrderNumber);
```

SQL CREATE INDEX Statement

- The DROP INDEX statement is used to delete an index in a table. :

ALTER TABLE *table_name*
DROP INDEX *index_name*;

Example:

ALTER TABLE Orders104
DROP INDEX idx_orderNumber;

SQL AUTO INCREMENT Field

- Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.

- Often this is the primary key field that we would like to be created automatically every time a new record is inserted.

SQL AUTO INCREMENT Field

- The following SQL statement defines the "ID" column to be an auto-increment primary key field in the "Persons105" table:

```
CREATE TABLE Persons105 (
    ID int NOT NULL AUTO_INCREMENT,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

SQL AUTO INCREMENT Field

- By default, the starting value for AUTO_INCREMENT is 1, and it will increment by 1 for each new record.
- To let the AUTO_INCREMENT sequence start with another value, use the following SQL statement:

```
ALTER TABLE Persons105 AUTO_INCREMENT=100;
```

SQL AUTO INCREMENT Field

- To insert a new record into the "Persons105" table, we will NOT have to specify a value for the "ID" column (a unique value will be added automatically). For example:

```
INSERT INTO Persons105 (FirstName,LastName)
VALUES (‘Fred’, 'Hammond’);
```

- You could verify this by displaying all the table’s contents using the following statement:

```
SELECT * FROM Persons105;
```

SQL Views

- In SQL, a view is a virtual table based on the result-set of an SQL statement.
- A view contains rows and columns, just like a real table.
- The fields in a view are fields from one or more real tables in the database.
- You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

SQL Views

- SQL Syntax for creating a view:

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

SQL Views

- A view always shows up-to-date data!
- The database engine recreates the data, using the view's SQL statement, every time a user queries a view.
- SQL Syntax for creating a view:

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

SQL Views

- You can run queries on these virtually created tables or even update the view to fetch specified data items from the table .
- When you are done using the view, you can drop it using the following syntax:

```
DROP VIEW view_name;
```

SQL GRANT and REVOKE Commands

- DCL commands are used to enforce database security in a multiple user database environment.
- Two types of DCL commands are GRANT and REVOKE.
- Only Database Administrator's or owner's of the database object can provide/remove privileges on a database object.

SQL GRANT and REVOKE Commands

- SQL GRANT is a command used to provide access or privileges on the database objects to the users.
- The Syntax is:

```
GRANT privilege,[privilege],.. ON privilege_level
TO user [IDENTIFIED BY password]
[REQUIRE tsl_option]
[WITH [GRANT_OPTION | resource_option]];
```

SQL GRANT and REVOKE Commands

- First, specify one or more privileges after the GRANT keyword.

- If you grant the user multiple privileges, each privilege is separated by a comma. (such as CREATE, DELETE, DROP, EXECUTE, etc).

SQL GRANT and REVOKE Commands

- Next, specify the privilege_level that determines the level at which the privileges apply.
- MySQL supports global (*.*), database (database.*), table (database.table) and column levels.
- If you use column privilege level, you must specify one or a list of comma-separated column after each privilege.

SQL GRANT and REVOKE Commands

- Then, place the user that you want to grant privileges. If user already exists, the GRANT statement modifies its privilege.
- Otherwise, the GRANT statement creates a new user.
- The optional clause IDENTIFIED BY allows you set a new password for the user.

SQL GRANT and REVOKE Commands

- After that, you specify whether the user has to connect to the database server over a secure connection such as SSL, X059, etc.

- Finally, the optional WITH GRANT OPTION clause allows you to grant other users or remove from other users the privileges that you possess.

SQL GRANT and REVOKE Commands

- In addition, you can use the WITH clause to allocate MySQL database server's resource e.g., to set how many connections or statements that the user can use per hour.
- This is very helpful in the shared environments such as MySQL shared hosting.

SQL GRANT and REVOKE Commands

- To demonstrate all that has been explained, let's start by creating a user.
- Typically, we use the CREATE USER statement to create a new user account first and then use the GRANT statement to grant privileges to the user.
- For example, the following CREATE USER statement creates a new user account for Sonia:

```
CREATE USER sonia@localhost IDENTIFIED BY 'soniapassword';
```

SQL GRANT and REVOKE Commands

- To display the privileges assigned to sonia@localhost user, you use SHOW GRANTS statement.

SHOW GRANTS FOR sonia@localhost;

SQL GRANT and REVOKE Commands

- To grant all privileges to the sonia@localhost user account, you use the following statement.

```
GRANT ALL ON *.* TO ‘sonia'@'localhost' WITH GRANT OPTION;
```

- The *.* shows that Sonia has privileges to all objects in all databases on the localhost system. For other levels, refer to previous slides.

SQL GRANT and REVOKE Commands

- The WITH GRANT OPTION allows sonia@localhost to grant privileges of the same kind or a subset of what she has to other users.

- Now if you use the SHOW GRANTS statement again, you will see that the privileges of sonia@localhost have been updated.

SQL GRANT and REVOKE Commands

- To create a user that has all privileges on a database by name “student101”, you use the following statements:

```
CREATE USER auditor@localhost IDENTIFIED BY 'whale';
GRANT ALL ON student101.* TO auditor@localhost;
```

SQL GRANT and REVOKE Commands

- You can grant multiple privileges in a single GRANT statement.
- For example, you can create a user that can execute the SELECT, INSERT and UPDATE statements against the “student101” database using the following statements:

```
CREATE USER rfc IDENTIFIED BY 'shark';
GRANT SELECT, UPDATE, DELETE ON student101.*
TO rfc;
```

SQL GRANT and REVOKE Commands

- To revoke any or all of the privileges granted a user or group of users, use the following revoke syntax:

```
REVOKE privilege_name
ON object_name
FROM {user_name |PUBLIC |role_name};
```

SQL GRANT and REVOKE Commands

- For example:

REVOKE SELECT ON student101.* FROM rfc;

- This command will REVOKE a SELECT privilege on all objects from the sudent101 database from rfc.
- When you REVOKE SELECT privilege on an object from a user, the user will not be able to SELECT data from these objects anymore.

SQL GRANT and REVOKE Commands

- However, if the user has received SELECT privileges on that object from more than one users, he/she can SELECT from that object until everyone who granted the permission revokes it.
- You cannot REVOKE privileges if they were not initially granted by you.

SQL GRANT and REVOKE Commands

- It's easier to GRANT or REVOKE privileges to the users through a role rather than assigning a privilege directly to every user.

- If a role is identified by a password, then, when you GRANT or REVOKE privileges to the role, you definitely have to identify it with the password.

SQL GRANT and REVOKE Commands

- To grant CREATE TABLE privilege to a user by creating a “testing” role:
- First, create a “testing” Role

```
CREATE ROLE testing;
```

- Second, grant a CREATE TABLE privilege to the role “testing”. You can add more privileges to the ROLE.

```
GRANT CREATE TABLE TO testing;
```

SQL GRANT and REVOKE Commands

- Third, grant the role to a user.

```
GRANT testing TO user1;
```

- To revoke a CREATE TABLE privilege from "testing" role, you can write:

```
REVOKE CREATE TABLE FROM testing;
```

SQL GRANT and REVOKE Commands

- The Syntax to drop a role from the database is as below:

```
DROP ROLE role_name;
```

- **For example:** To drop a role called testing, you can write:

```
DROP ROLE testing;
```

END

THANKS!

Any questions?

You can find me at elielkeelson@gmail.com &
ekeelson@knust.edu.gh