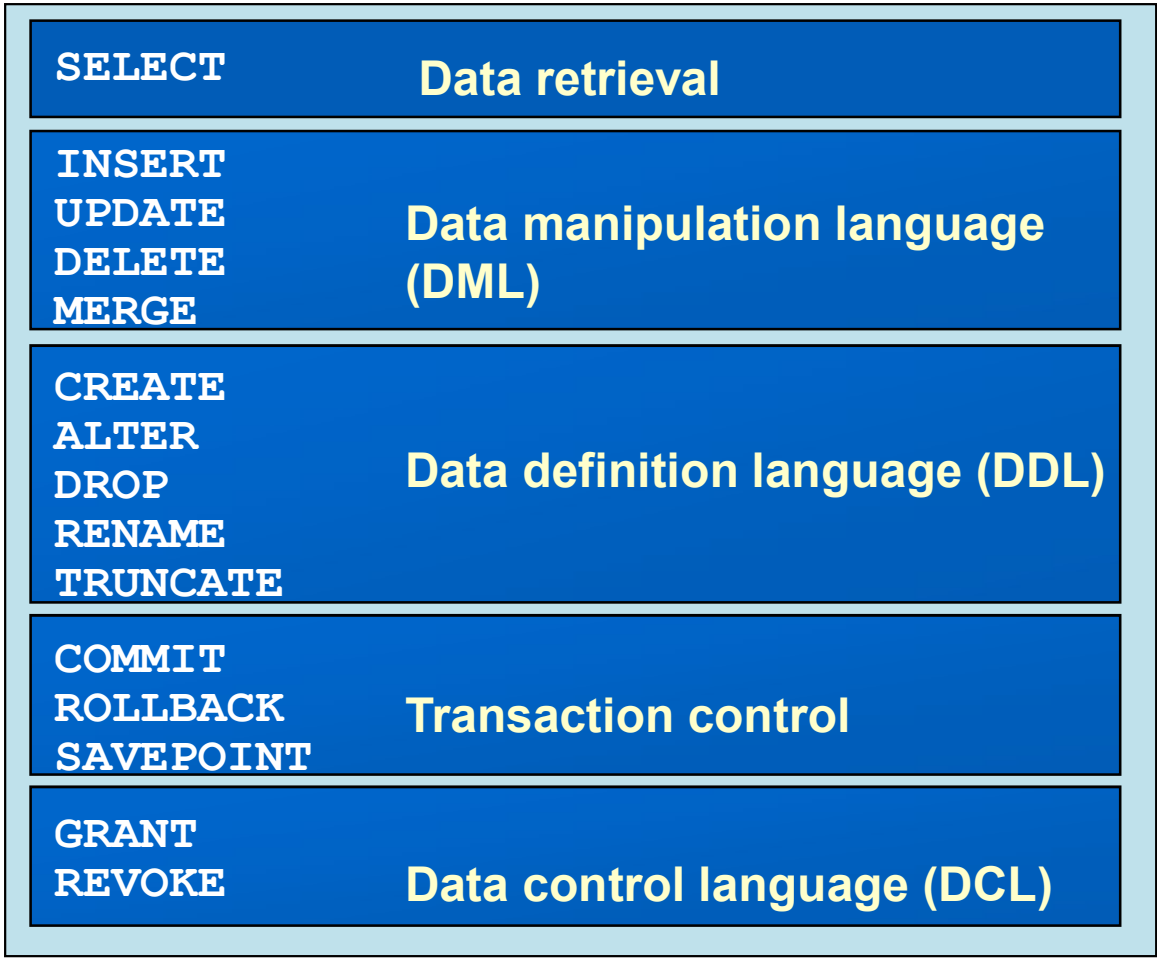
SQL DDL DML COMMANDS

History

- IBM Sequel language developed as part of System R project at the IBM San Jose Research Laboratory
- Renamed Structured Query Language (SQL)
- ANSI and ISO standard SQL:
  - SQL-86
  - SQL-89
  - SQL-92
  - SQL:1999 (language name became Y2K compliant!)
  - SQL:2003
- Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.

SQL Statments



Most Important SQL Commands

- SELECT - extracts data from a database
- UPDATE - updates data in a database
- DELETE - deletes data from a database
- INSERT INTO - inserts new data into a database
- CREATE DATABASE - creates a new database
- ALTER DATABASE - modifies a database
- CREATE TABLE - creates a new table
- ALTER TABLE - modifies a table
- DROP TABLE - deletes a table
- CREATE INDEX - creates an index (search key)
- DROP INDEX - deletes an index

SQL Data Types

Data type	Description	Max size	Storage
char(n)	Fixed width character string	8,000 characters	Defined width
varchar(n)	Variable width character string	8,000 characters	2 bytes + number of chars
varchar(max)	Variable width character string	1,073,741,824 characters	2 bytes + number of chars
text	Variable width character string	2GB of text data	4 bytes + number of chars
nchar	Fixed width Unicode string	4,000 characters	Defined width x 2
nvarchar	Variable width Unicode string	4,000 characters	
nvarchar(max)	Variable width Unicode string	536,870,912 characters	
ntext	Variable width Unicode string	2GB of text data	
binary(n)	Fixed width binary string	8,000 bytes	
varbinary	Variable width binary string	8,000 bytes	
varbinary(max)	Variable width binary string	2GB	
image	Variable width binary string	2GB	

Table creation

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

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

Create Table Using Another Table

- A copy of an existing table can also be created using CREATE TABLE.
- 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.

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

SQL INSERT INTO Statement

- The INSERT INTO statement is used to insert new records in a table.

INSERT INTO Syntax

- It is possible to write the INSERT INTO statement in two ways.

- The first way specifies both the column names and the values to be inserted:

INSERT INTO *table_name* (*column1*, *column2*, *column3*, ...)
VALUES (*value1*, *value2*, *value3*, ...);

- If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. The INSERT INTO syntax would be as follows:

INSERT INTO *table_name*
VALUES (*value1*, *value2*, *value3*, ...);

SQL DROP TABLE Statement

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

Syntax

```
DROP TABLE table_name;
```

SQL TRUNCATE TABLE

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

```
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 is also used to add and drop various constraints on an existing table.

ALTER TABLE - ADD Column

- To add a column in a table, use the following syntax:

```
ALTER TABLE table_name
ADD column_name datatype;
```

ALTER TABLE - DROP COLUMN

To delete a column in a table

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

SQL ALTER TABLE Statement

- ALTER TABLE - ALTER/MODIFY COLUMN

```
ALTER TABLE table_name
ALTER COLUMN column_name
datatype;
```

SQL Create 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.

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

- 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.

SQL Create Constraints

- **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** - Uniquely identifies a row/record in another table
- **CHECK** - Ensures that all values in a column satisfies a specific condition
- **DEFAULT** - Sets a default value for a column when no value is specified
- **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 on CREATE TABLE

- The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values when the "Persons" table is created:
- CREATE TABLE Persons (

  ID int NOT NULL,

  LastName varchar(255) NOT NULL,

  FirstName varchar(255) NOT NULL,

  Age int

  );

SQL NOT NULL Constraint

- To create a NOT NULL constraint on the "Age" column when the "Persons" table is already created, use the following SQL:

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

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.

SQL UNIQUE Constraint on CREATE TABLE

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

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).

SQL PRIMARY KEY on CREATE TABLE

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

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

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

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

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

Note : If you use the ALTER TABLE statement to add a primary key, the primary key column(s) must already have been declared to not contain NULL values (when the table was first created).

SQL PRIMARY KEY Constraint

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

```
ALTER TABLE Persons
DROP CONSTRAINT PK_Person;
```

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

- 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 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.

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

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

SQL FOREIGN KEY Constraint

- SQL FOREIGN KEY on CREATE TABLE
- 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)

);

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

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 on CREATE TABLE

- The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that the age of a person must be 18, or older:
- CREATE TABLE Persons (
  ID int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Age int,
  CHECK (Age>=18)
  );
- 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 on ALTER TABLE

- 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);
```

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='Sandnes');
```

DROP a CHECK Constraint

- ALTER TABLE Persons
  DROP CONSTRAINT CHK_PersonAge;

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.

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

SQL DEFAULT Constraint

SQL DEFAULT on ALTER TABLE

To create a DEFAULT constraint on the "City" column when the table is already created, use the following SQL:

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

SQL SELECT Statement

- The SELECT statement is used to select data from a database.
- The data returned is stored in a result table, called the result-set.

Syntax:

SELECT *column1*, *column2*, ...
FROM *table_name*;

To select all the fields available in the table, use the following syntax:

SELECT * FROM *table_name*;

SQL SELECT DISTINCT Statement

- The SELECT DISTINCT statement is used to return only distinct (different) values.

Syntax:

SELECT DISTINCT *column1*, *column2*, ...
FROM *table_name*;

SQL Where clause

- The WHERE clause is used to filter records.
- The WHERE clause is used to extract only those records that fulfill a specified condition.

Syntax:

SELECT *column1, column2, ...*
FROM *table_name*
WHERE *condition*;

Operators in The WHERE Clause

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
BETWEEN	Between a certain range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

SQL AND, OR and NOT Operators

- The WHERE clause can be combined with AND, OR, and NOT operators.
- The AND and OR operators are used to filter records based on more than one condition:
  - ✓ The AND operator displays a record if all the conditions separated by AND are TRUE.
  - ✓ The OR operator displays a record if any of the conditions separated by OR is TRUE.
- The NOT operator displays a record if the condition(s) is NOT TRUE.

SQL AND, OR and NOT Operators

AND Syntax

SELECT *column1*, *column2*, ...
FROM *table_name*
WHERE *condition1* AND *condition2* AND *condition3* ...;

```
SELECT * FROM Customers
WHERE Country='Germany' AND City='Berlin';
```

OR Syntax

SELECT *column1*, *column2*, ...
FROM *table_name*
WHERE *condition1* OR *condition2* OR *condition3* ...;

```
SELECT * FROM Customers
WHERE City='Berlin' OR City='München';
```

NOT Syntax

SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;

```
SELECT * FROM Customers
WHERE NOT Country='Germany';
```

SQL ORDER BY

- The ORDER BY keyword is used to sort the result-set in ascending or descending order.
- The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

ORDER BY Syntax

- SELECT *column1*, *column2*, ...
  FROM *table_name*
  ORDER BY *column1*, *column2*, ... ASC|DESC;
-

```
SELECT * FROM Customers
ORDER BY Country;
```

```
SELECT * FROM Customers
ORDER BY Country DESC;
```

```
SELECT * FROM Customers
ORDER BY Country, CustomerName;
```

```
SELECT * FROM Customers
ORDER BY Country ASC,
CustomerName DESC;
```

SQL NULL Values

- A field with a NULL value is a field with no value.
- If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.

Note: A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation!

IS NULL Syntax

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL;
```

IS NOT NULL Syntax

```
SELECT column_names
FROM table_name
WHERE column_name IS NOT NULL;
```

SQL UPDATE Statement

- The UPDATE statement is used to modify the existing records in a table.

UPDATE Syntax

- UPDATE *table_name*

  SET *column1* = *value1*, *column2* = *value2*, ...

  WHERE *condition*;

Note: The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated!

```
UPDATE Customers
SET ContactName = 'Alfred
Schmidt', City= 'Frankfurt'
WHERE CustomerID = 1;
```

SQL DELETE Statement

- The DELETE statement is used to delete existing records in a table.

DELETE Syntax

DELETE FROM *table_name* WHERE *condition*;

Note: The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted!

Delete All Records
It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

DELETE FROM *table_name*;

```
DELETE FROM Customers WHERE
CustomerName='Alfreds  Futterkiste';
```

SQL MIN() and MAX() Functions

The SQL MIN() and MAX() Functions

- The MIN() function returns the smallest value of the selected column.
- The MAX() function returns the largest value of the selected column.

MIN() Syntax

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

MAX() Syntax

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

SQL MIN() and MAX() Functions

ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35

```
SELECT MIN(Price) AS SmallestPrice
FROM Products;
```

```
SELECT MAX(Price) AS LargestPrice
FROM Products;
```

SQL COUNT(), AVG() and SUM() Functions

- The COUNT() function returns the number of rows that matches a specified criterion.
- The AVG() function returns the average value of a numeric column.
- The SUM() function returns the total sum of a numeric column.

COUNT() Syntax

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

AVG() Syntax

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

SUM() Syntax

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

SQL COUNT(), AVG() and SUM() Functions

ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35

```
SELECT COUNT(ProductID) FROM Products;

SELECT AVG(Price) FROM Products;

SELECT SUM(Price)FROM Products;
```