FØRSTE OBLIGATORISKE OPPGAVE

DATABASER

Amna Dastgir (s364520)

Informasjonsteknologi

Innholdsfortegnelse

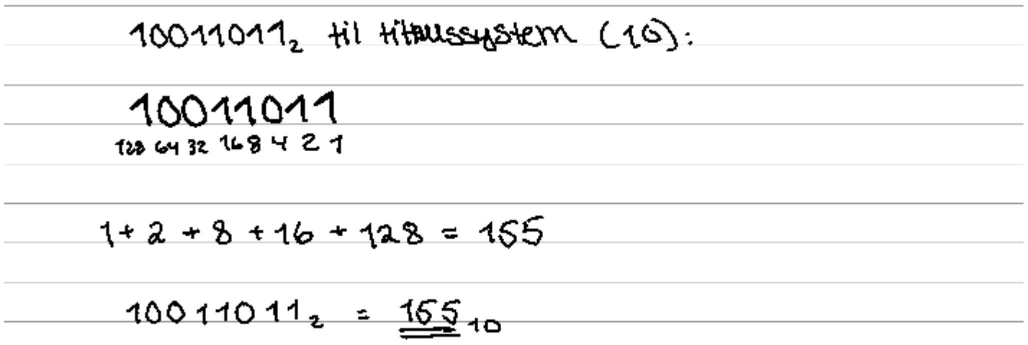
DATABASER OBLIGATORISKE OPPGAVE 1

FORORD:

Mye i denne obligatoriske oppgaven har W3Schools som kilde og er dermed kildehenvist til med APA-stils bibliografi. Det er ikke hensikt i å ta ære for ord eller kunnskap, og den såkalte «læreboka» som er skrevet i oppgavesett en og to baserer seg sterkt på kilden.

OPPGAVESETT 1

Regn om 10011011 fra binært til 10-tallsystem:



10011011_2 til titallssystem (10):

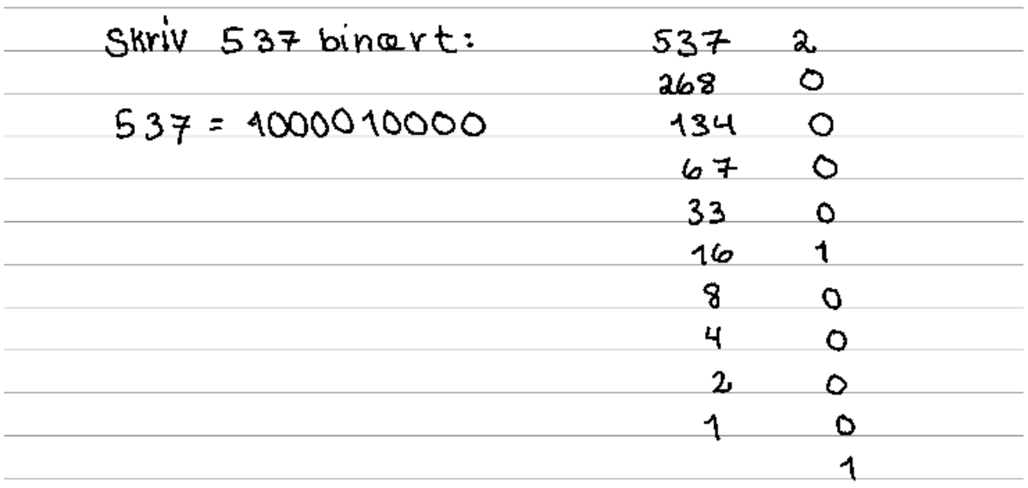
10011011

128 64 32 16 8 4 2 1

$1 + 2 + 8 + 16 + 128 = 155$

$10011011_2 = \underline{\underline{155}}_{10}$

Skriv 537 binært:

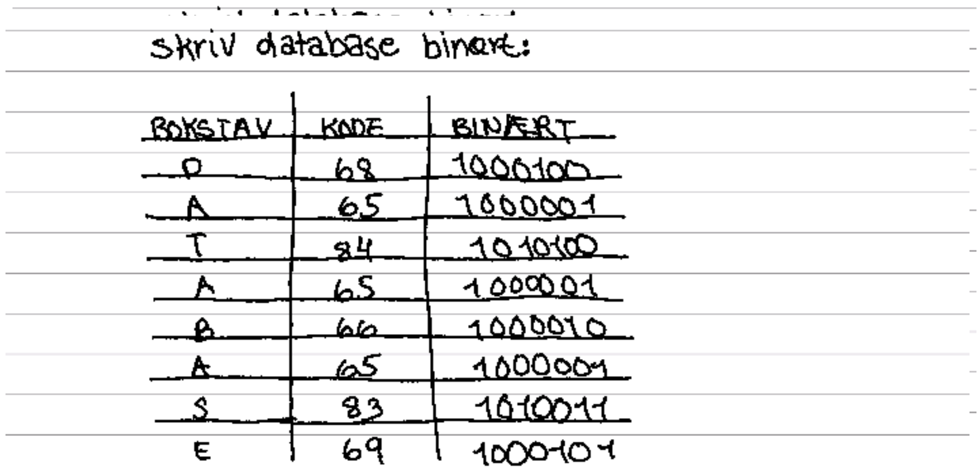


Skriv 537 binært:

537 = 1000010000

537	2
268	0
134	0
67	0
33	0
16	1
8	0
4	0
2	0
1	0
	1

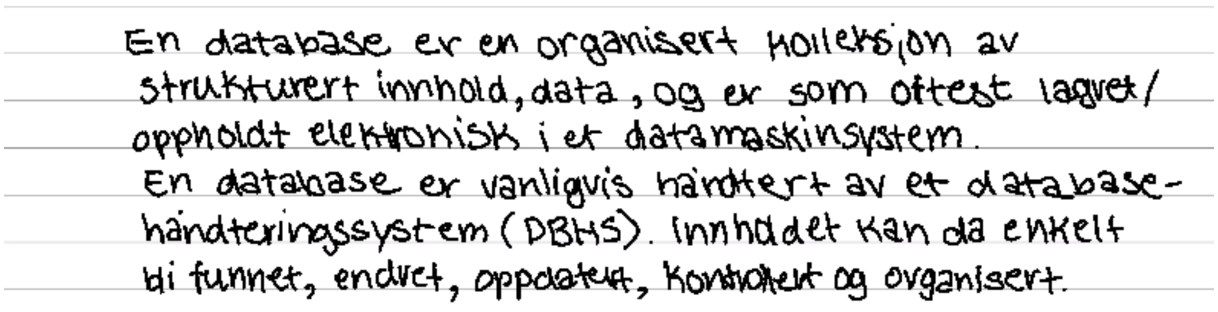
Skriv DATABASE binært:



skriv database binært:

BOKSTAV	KODE	BINÆRT
D	68	1000100
A	65	1000001
T	84	1010100
A	65	1000001
B	66	1000010
A	65	1000001
S	83	1010011
E	69	1000101

Hva er en database?



En database er en organisert kolleksjon av
strukturert innhold, data, og er som oftest lagret/
oppholdt elektronisk i et datamaskinsystem.
En database er vanligvis håndtert av et database-
håndteringssystem (DBHS). Innholdet kan da enkelt
bli funnet, endret, oppdatert, kontrollert og organisert.

Hva er en relasjonsdatabase?

En relasjonsdatabase består av tabeller som har relasjoner mellom seg. Relasjonsdatabase er en database som bygger seg på relasjonsmodellen.

SQL LÆREFORTEGNELSE

SQL STATEMENT	IS USED FOR	EXAMPLE

SELECT	Is used to select data from a database. Data returned is stored in a result table, called result-set (W3Schools, 1999-2022).	SELECT *column1, column2*, …
		FROM *table_name*;
		To select all the fields:
		SELECT * FROM *table_name*;

SELECT DISTINCT	Is used to return only distinct (different) values.	SELECT DISTINCT *column1, column2*, …
		FROM *table_name*;

WHERE	Is used to filter records. Is used to extract only those records that fulfill a specified condition.	SELECT *column1, column2, …*
		FROM *table_name*
		WHERE *Country = 'Mexico';*

AND, OR and NOT	WHERE can be combined with these. These are used in literal terms.	`SELECT column1, column2, ...`
		`FROM table_name`
		`WHERE condition1 AND condition2 AND condition3 ...;`
		`WHERE condition1 OR condition2 OR condition3 ...;`
		`WHERE NOT condition;`

ORDER BY	Is used to sort the result-set in ascending or descending order. To sort the record in descending order use DESC.	`SELECT column1, column2, ...`
		`FROM table_name`
		`ORDER BY column1, column2, ... ASC|DESC;`

INSERT INTO	Is used to insert new records in a table. This statement can be written in two ways:	`INSERT INTO table_name (column1, column2, column3, ...) VALUES (value1, value2, value3, ...);`
		
		 `INSERT INTO table_name VALUES (value1, value2, value3, ...);`

NULL VALUES	A field with NULL value is a field with no value.	`SELECT column_names FROM table_name WHERE column_name IS NULL;`
		
		 `SELECT column_names FROM table_name WHERE column_name IS NOT NULL;`

UPDATE	Is used to modify the existing records in a table.	`UPDATE table_name SET column1 = value1, column2 = value2, ... WHERE condition;`
DELETE	Is used to delete existing records in a table. It is important to write the WHERE clause correct.	`DELETE FROM table_name WHERE condition;`

SELECT TOP/ LIMIT	Is used to specify the number of records to return. MySQL supports LIMIT instead of SELECT TOP.	`SELECT TOP 3 * FROM Customers;`
		
		 For MySQL:
		 `SELECT * FROM Customers LIMIT 3;`

MIN() AND MAX()	The MIN() function returns the smallest value of the selected column, where the MAX() does the opposite.	`SELECT MIN(column_name)`
		`FROM table_name`
		`WHERE condition;`
		
		`SELECT MAX(column_name)`
		`FROM table_name`
		`WHERE condition;`

COUNT( ), AVG(), SUM()	COUNT() return the number of rows that matched a specific criteria.
	The AVG() function returns the average value of a numeric column.
	The SUM() function returns the total sum of a numeric column
		`SELECT COUNT(column_name)`
		`FROM table_name`
		`WHERE condition;`
		
		`SELECT AVG(column_name)`
		`FROM table_name`
		`WHERE condition;`
		
		`SELECT SUM(column_name)`
		`FROM table_name`
		`WHERE condition;`

LIKE	This operator is used in a WHERE clause to search for a specified pattern in a column.	Selects everyone who starts with an “a”:
		`SELECT * FROM Customers`
		`WHERE CustomerName LIKE 'a%';`
		
		Selects everyone whos name ends with an “a”:
		`SELECT * FROM Customers`
		`WHERE CustomerName LIKE '%a';`
		
		Selects all customers with a CustomerName that have "or" in any position:
		`SELECT * FROM Customers`
		`WHERE CustomerName LIKE '%or%';`

		Selects with “r” in second position:
		`SELECT * FROM Customers`
		`WHERE CustomerName LIKE '_r%';`
		
		Starts with "a" and are at least 3 characters in length:
		`SELECT * FROM Customers`
		`WHERE CustomerName LIKE 'a__%';`
		
		Selects everyone who does not start with an “a”:
		`SELECT * FROM Customers`
		`WHERE CustomerName NOT LIKE 'a%'`
		`;`

WILDCARDS	Is used to substitute one or more characters in a string.
	
	Wildcard characters are used with the LIKE operator.
	
	The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
		Selects all customers with a City starting with "ber":
		`SELECT * FROM Customers`
		`WHERE City LIKE 'ber%';`
		
		Customers with a City containing the pattern "es":
		`SELECT * FROM Customers`
		`WHERE City LIKE '%es%';`

IN	The IN operator allows you to specify multiple values in a WHERE clause.
	
	The IN operator is a shorthand for multiple OR conditions.
		`SELECT column_name(s)`
		`FROM table_name`
		`WHERE column_name IN (value1, va`
		`lue2, ...);`
		
		`SELECT column_name(s)`
		`FROM table_name`

		`WHERE column_name IN (SELECT STA TEMENT);`

BETWEEN	The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.
	
	The BETWEEN operator is inclusive: begin and end values are included.
		`SELECT column_name(s)`
		`FROM table_name`
		`WHERE column_name BETWEEN value1 AND value2;`

ALIASES	Used to give a table, or a column in a table, a temporary name. An alias is created with the AS keyword.	Column:
		`SELECT Column_name AS alias_name`
		`FROM table_name;`
		
		Table:
		`SELECT column_name(s)`
		`FROM table_name AS alias_name;`

JOINS	A JOIN clause is used to combine rows from two or more tables, based on a related column between them.	`S`
		
		`ELECT Orders.OrderID, Customers.CustomerName,Orders.OrderDate`
		`FROM Orders`
		`INNER JOIN Customers ON Orders.CustomerID =Customers.CustomerID;`

INNER JOIN	The INNER JOIN keyword selects records that have matching values in both tables.	`SELECT column_name(s)`
		`FROM table1`
		`INNER JOIN table2`
		`ON table1.column_name = table2.column_name;`

LEFT JOIN	Returns all records from the left table (table1), and the matching records from the right table (table2).	`SELECT column_name(s)`
		`FROM table1`
		`LEFT JOIN table2`
		`ON table1.column_name = table2.column_name;`

RIGHT JOIN	returns all records from the right table (table2), and the matching records from the left table (table1).	`SELECT column_name(s)`
		`FROM table1`
		`RIGHT JOIN table2`
		`ON table1.column_name = table2.column_name;`

FULL JOIN	The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records. FULL OUTER JOIN and FULL JOIN are the same.	

		`SELECT column_name(s)`
		`FROM table1`
		`FULL OUTER JOIN table2`
		`ON table1.column_name = table2.c`
		`olumn_name`
		`WHERE condition;`

SELF JOIN	A self join is a regular join, but the table is joined with itself.	`SELECT column_name(s)`
		`FROM table1 T1, table1 T2`
		`WHERE condition;`

UNION	The UNION operator is used to combine the result-set of two or more SELECT statements.
	
	Every SELECT statement within UNION must have the same number of columns
	The columns must also have similar data types
	The columns in every SELECT statement must also be in the same order
		`SELECT column_name(s) FROM table`
		`1`
		`UNION`
		`SELECT column_name(s) FROM table`
		`2;`
		
		For duplicate values also:
		`SELECT column_name(s) FROM table`
		`1`
		`UNION ALL`
		`SELECT column_name(s) FROM table`
		`2;`

	The GROUP BY statement groups rows that have the same values into summary rows, like	

GROUP BY	"find the number of customers in each country".
	
	The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.
		`SELECT column_name(s)`
		`FROM table_name`
		`WHERE condition`
		`GROUP BY column_name(s)`
		`ORDER BY column_name(s);`

HAVING	The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.	`SELECT column_name(s)`
		`FROM table_name`
		`WHERE condition`
		`GROUP BY column_name(s)`
		`HAVING condition`
		`ORDER BY column_name(s);`

EXISTS	The EXISTS operator is used to test for the existence of any record in a subquery.
	
	The EXISTS operator returns TRUE if the subquery returns one or more records.
		`SELECT column_name(s)`
		`FROM table_name`
		`WHERE EXISTS`
		`(SELECT column_name FROM table_n`
		`ame WHERE condition);`

	The ANY and ALL operators allow you to perform a comparison between a single column value and a range of other values.
	The ANY operator:
		ANY Syntax:
		`SELECT column_name(s)`
		`FROM table_name`
		`WHERE column_name operator ANY`
		`  (SELECT column_name`

ANY, ALL	returns a boolean value as a result
	returns TRUE if ANY of the subquery values meet the condition
	ANY means that the condition will be true if the operation is true for any of the values in the range.
	
	The ALL operator:
	• returns a boolean value as a result
	• returns TRUE if ALL of the subquery values meet the condition
	• is used with SELECT, WHERE an d HAVING statements
	
	ALL means that the condition will be true only if the operation is true for all values in the range.
		`FROM table_name`
		`WHERE condition);`
		
		ALL Syntax:
		`SELECT ALL column_name(s)`
		`FROM table_name`
		`WHERE condition;`

SELECT INTO	The SELECT INTO statement copies data from one table into a new one.	Copies all:
		`SELECT *`
		`INTO newtable [IN externaldb]`
		`FROM oldtable`
		`WHERE condition;`
		
		Copies some:
		`SELECT column1, column2, column3, ...`
		`INTO newtable [IN externaldb]`

		`FROM oldtable`
		`WHERE condition;`

INSERT INTO SELECT	The INSERT INTO SELECT statement copies data from one table and inserts it into another table.
	
	The INSERT INTO SELECT statement requires that the data types in source and target tables match.
		Copies all columns:
		`INSERT INTO table2`
		`SELECT * FROM table1`
		`WHERE condition;`
		
		Copies some columns:
		`INSERT INTO table2 (column1, col`
		`umn2, column3, ...)`
		`SELECT column1, column2, column3`
		`, ...`
		`FROM table1`
		`WHERE condition;`

CASE	The CASE statement goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.
	
	If there is no ELSE part and no conditions are true, it returns NULL.
		`CASE`
		`    WHEN condition1 THEN result1`
		`    WHEN condition2 THEN result2`
		`    WHEN conditionN THEN resultN`
		`    ELSE result`
		`END;`

NULL FUNCTIONS	SQL IFNULL(), ISNULL(), COALESCE(), and NVL() Functions	Other functions with different queries

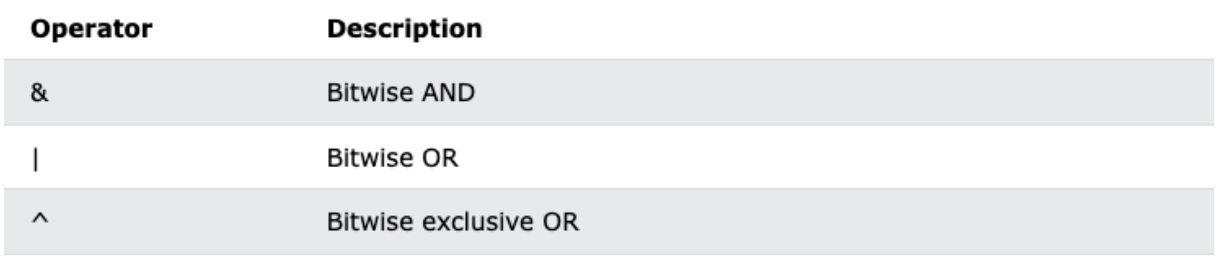
STORED PROCEDURES	A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.
	
	So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.
	
	You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.
		`CREATE PROCEDURE procedure_name`
		`AS`
		`sql_statement`
		`GO;`

COMMENTS	Single line comments start with --.	`--Select all:`
		`SELECT * FROM Customers;`

SQL Arithmetic Operators

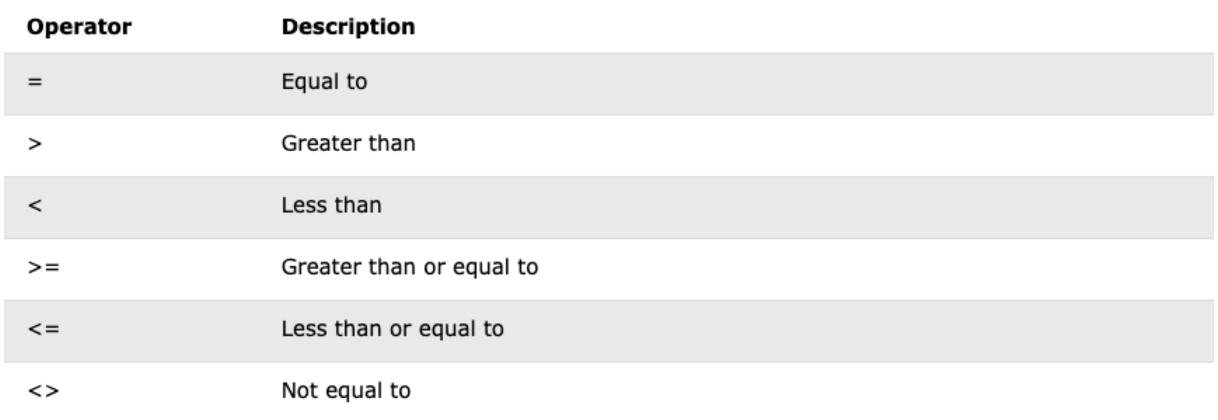
Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulo

SQL Bitwise Operators



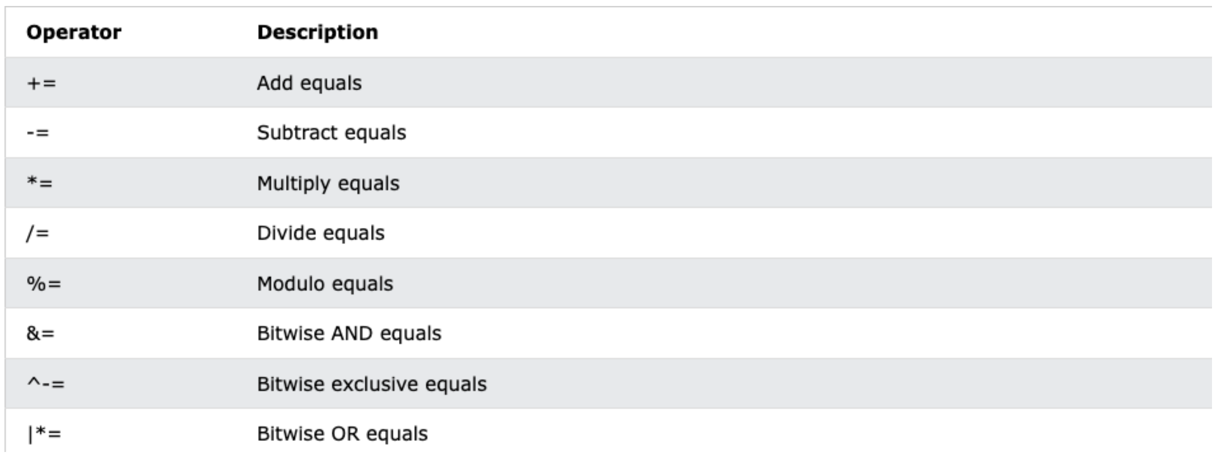
Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR

SQL Comparison Operators



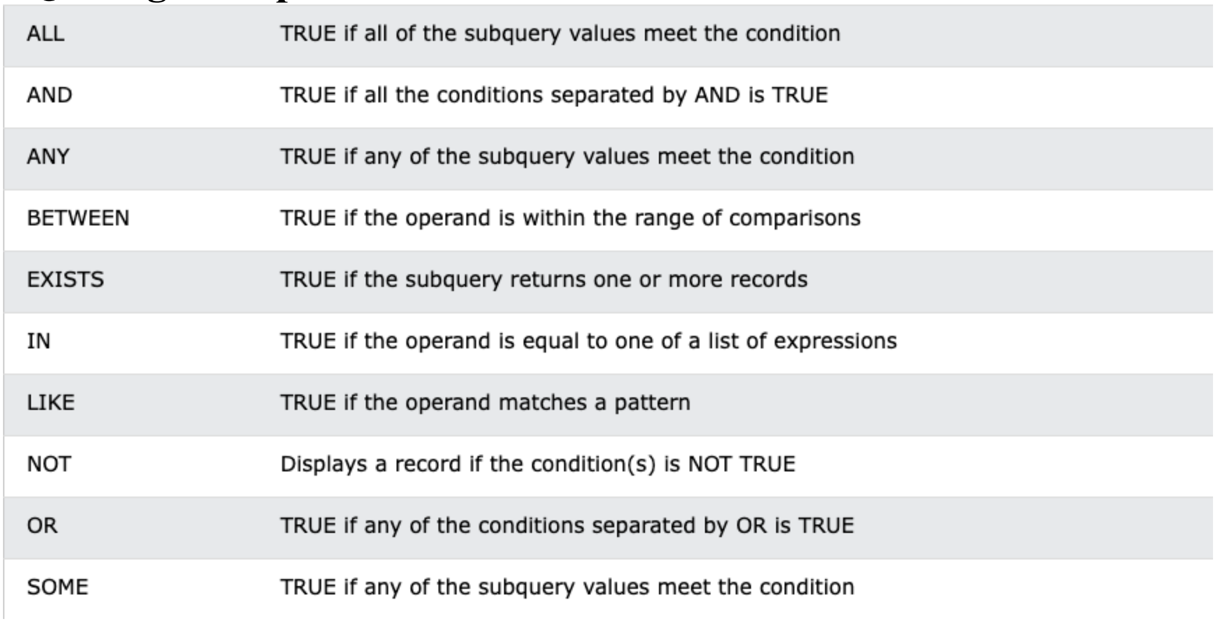
Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

SQL Compound Operators



Operator	Description
+=	Add equals
-=	Subtract equals
*=	Multiply equals
/=	Divide equals
%=	Modulo equals
&=	Bitwise AND equals
^-=	Bitwise exclusive equals
|*=	Bitwise OR equals

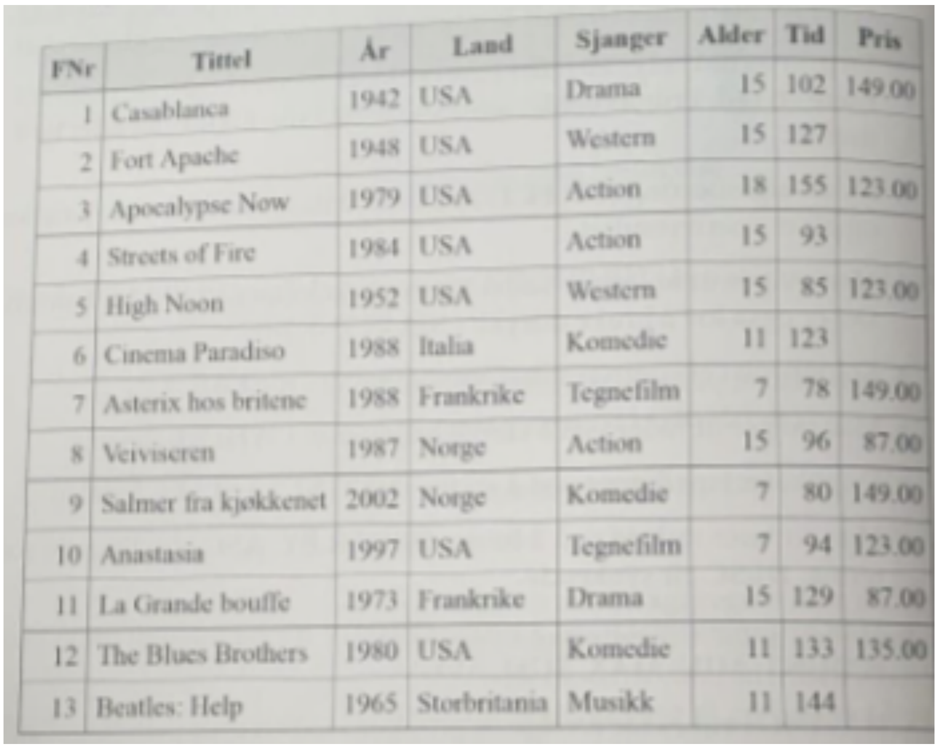
SQL Logical Operators



ALL	TRUE if all of the subquery values meet the condition
AND	TRUE if all the conditions separated by AND is TRUE
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records
IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions separated by OR is TRUE
SOME	TRUE if any of the subquery values meet the condition

FILMOPPGAVE KAPITTEL TO

Mediebedriften BlancaFlix har spesialisert seg på strømming og nedlastning av eldre filmklassikere. Tabellen Film vist i figur 2.16 inneholder data om de filmene butikken fører FNr inneholder et unikt filmnummer og er valgt som primærnøkkel. De øvrige kolonnene inneholder filmtittel, produksjonsår, produksjonsland, sjanger, aldersgrense, spilletid i minutter og nedlastningspris. Merk at noen filmer ikke er tilgjengelig for nedlastning og har derfor nullmerke i kolonnen Pris (de er kun tilgjengelig i strømmetjenesten via en abonnementsordning).
Skriv SQL-spørringer for å vise følgende opplysninger fra filmtabellen:



FNr	Tittel	År	Land	Sjanger	Alder	Tid	Pris
1	Casablanca	1942	USA	Drama	15	102	149.00
2	Fort Apache	1948	USA	Western	15	127	
3	Apocalypse Now	1979	USA	Action	18	155	123.00
4	Streets of Fire	1984	USA	Action	15	93	
5	High Noon	1952	USA	Western	15	85	123.00
6	Cinema Paradiso	1988	Italia	Komedie	11	123	
7	Asterix hos britene	1988	Frankrike	Tegnefilm	7	78	149.00
8	Veiviseren	1987	Norge	Action	15	96	87.00
9	Salmer fra kjøkkenet	2002	Norge	Komedie	7	80	149.00
10	Anastasia	1997	USA	Tegnefilm	7	94	123.00
11	La Grande bouffe	1973	Frankrike	Drama	15	129	87.00
12	The Blues Brothers	1980	USA	Komedie	11	133	135.00
13	Beatles: Help	1965	Storbritania	Musikk	11	144	

Skriv SQL-Spørringer for å vise følgende opplysninger fra filmtabellen:

a) **All informasjon om filmer produsert i 1988:**

```
SELECT *
FROM Film
WHERE År = 1988
```

b) **Tittel på amerikanske filmer produsert på 1950-tallet:**

```
SELECT FNr, Tittel
FROM Film
WHERE Land = ‘USA’ AND År BETWEEN 1980 AND 1989
```

c) Komedier med aldersgrense under 10 år og spilletid under 130 minutter:

```
SELECT *
FROM Film
WHERE Sjanger = ‘Komedie’ AND Alder  < 10 AND Tid < 130
```

d) Tittel på alle action- og westernfilmer:

```
SELECT FNr, Tittel
FROM Film
WHERE Sjanger = ‘Action’ OR Sjanger = ‘Western’
```

e) Alle produksjonsland, sortert og uten gjentakelser:

```
SELECT DISTINCT Land
FROM Film
ORDER BY Land
```

f) Korteste og lengste spilletid innen hver sjanger:

```
SELECT Sjanger, MIN(Tid) AS Korteste, (MAX(Tid) AS Lengste
FROM Film
GROUP BY Sjanger
```

g) Antall filmer som ikke er til salgs:

```
SELECT COUNT(*) AS IkkeTilSalgs
FROM Film
WHERE Pris IS NULL
```

h) Antall filmer under 100 kr:

```
SELECT COUNT(Pris) AS FilmerUnder100Kr
FROM Film
WHERE Pris < 100
```

i) Filmer med tittel som slutter på 'now':

```
SELECT *
FROM Film
WHERE UPPER(Tittel LIKE '*NOW'
```

j) Gjennomsnittspris for sjangre med flere enn to filmer:

```
SELECT Sjanger, AVG(Pris) AS GjennomsnittsPris
FROM Film
GROUP BY Sjanger
HAVING COUNT(*) > 2
```

k) Differansen mellom dyreste og billigste film innen hver sjanger:

```
SELECT Sjanger, MAX(Pris)-MIN(Pris) AS Differanse
FROM Film
GROUP BY Sjanger
```

l) Totalt antall filmer og antall filmer til salgs, fordelt på produksjonsland:

```
SELECT Land, COUNT(*) AS TotaltAntall, COUNT(Pris) AS AntallTilSalgs
FROM Film
GROUP BY Land
```

m) Antall år siden utgivelse for filmer eldre enn 50 år. Tips: Vedlegg A lister navn på noen datofunksjoner. Du trenger kanskje en funksjon for å finne dagens dato, og en funksjon for å trekke ut årstallet fra en dato:

```
SELECT Fnr, YEAR(CURDATE())-År AS AntallÅr
FROM Film
WHERE YEAR(CURDATE())-ÅR > 50;
```

Oppgavesett 2

SQL STATEMENT	IS USED FOR	EXAMPLE
CREATE DATABASE	Is used to create a new SQL database.	`CREATE DATABASE databasename;`
DROP DATABASE	Is used to drop av an existing SQL database.	`DROP DATABASE databasename;`
BACKUP DATABASE	Is used in SQL server to create a full backup of an existing SQL database.	`BACKUP DATABASE databasename` `TO DISK = 'filepath';`

CREATE TABLE	Is used to create a new table in a database.	`CREATE TABLE table_name (` `column1 datatype,` `column2 datatype,` `column3 datatype,` `....` `);`
		
		 Example:
		 `CREATE TABLE TestTable AS` `SELECT customername, contactname` `FROM customers;`

DROP TABLE TRUNCATE TABLE	Is used to drop an existing table in a database. TRUNCATE TABLE is used to delete the data inside a table, but not the table itself.	`DROP TABLE table_name;`
		
		`TRUNCATE TABLE table_name;`

ALTER TABLE	Is used to add, delete, or modify columns in an existing. And also drop various constrains on an existing table.	`ALTER TABLE table_name`
		`ADD column_name datatype;`
		
		Adds "email" columns to the "customers" table:
		`ALTER TABLE Customers`
		`ADD Email varchar(255);`

CONSTRAINS	Used to specify rules for data in a table, and can be done when the table is created with CREATE TABLE or after with ALTER TABLE.	`CREATE TABLE table_name (`
		`    column1 datatype constraint,`
		`    column2 datatype constraint,`
		`    column3 datatype constraint,`
		`    ....`
		`);`
		
		• 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

NOT NULL	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`
		`);`

UNIQUE	The UNIQUE constraint ensures that all values in a column are different.	`CREATE TABLE Persons (`
		`    ID int NOT NULL UNIQUE,`
		`    LastName varchar(255) NOT NULL,`
		`    FirstName varchar(255),`
		`    Age int`
		`);`

	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.	

PRIMARY KEY	The PRIMARY KEY constraint uniquely identifies each record in a table.
	
	Primary keys must contain UNIQUE values, and cannot contain NULL values.
		`CREATE TABLE Persons (`
		`    ID int NOT NULL,`
		`    LastName varchar(255) NOT NULL,`
		`    FirstName varchar(255),`
		`    Age int,`
		`    PRIMARY KEY (ID)`
		`);`
		
		`CREATE TABLE Persons (`
		`    ID int NOT NULL,`
		`    LastName varchar(255) NOT NULL,`
		`    FirstName varchar(255),`
		`    Age int,`
		`    CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)`
		`);`

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

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

CHECK	The CHECK constraint is used to limit the value range	`CREATE TABLE Persons (`
		`    ID int NOT NULL,`
		`    LastName`
		`varchar(255) NOT NULL,`
		`    FirstName varchar(255),`
		`    Age int,`

	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.
		`    CHECK (Age>=18)`
		`);`
		
		`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')`
		`);`

DEFAULT	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'`
		`);`

INDEX	The CREATE INDEX statement is used to create indexes in tables.	`CREATE INDEX index_name`
		`ON table_name (column1, column2,`
		`...);`

	Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.	

AUTO INCREMENT	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.
		`CREATE TABLE Persons (`
		`    Personid int NOT NULL AUTO_INCREMENT,`
		`    LastName varchar(255) NOT NULL,`
		`    FirstName varchar(255),`
		`    Age int,`
		`    PRIMARY KEY (Personid)`
		`);`

DATES	The most difficult part when working with dates is to be sure that the format of the date you are trying to insert, matches the format of the date column in the database.	**MySQL** comes with the following data types for storing a date or a date/time value in the database: • DATE - format YYYY-MM-DD • DATETIME - format: YYYY-MM-DD HH:MI:SS • TIMESTAMP - format: YYYY-MM-DD HH:MI:SS • YEAR - format YYYY or YY **SQL Server** comes with the following data types for storing a

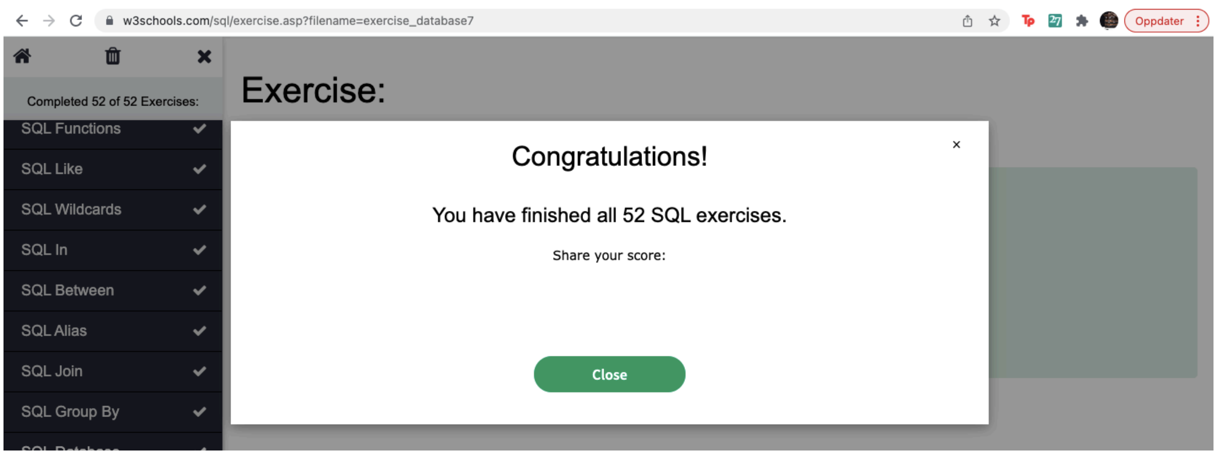
	As long as your data contains only the date portion, your queries will work as expected. However, if a time portion is involved, it gets more complicated.	date or a date/time value in the database: • DATE - format YYYY-MM-DD • DATETIME - format: YYYY-MM-DD HH:MI:SS • SMALLDATETIME - format: YYYY-MM-DD HH:MI:SS • TIMESTAMP - format: a unique number **Note:** The date types are chosen for a column when you create a new table in your database!

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 statements and functions to a view and present the data as if the data were coming from one single table.
		`CREATE VIEW view_name AS`
		`SELECT column1, column2, ...`
		`FROM table_name`
		`WHERE condition;`

	A view is created with the CREATE VIEW statement.	
INJECTION	SQL injection is a code injection technique that might destroy your database. SQL injection is one of the most common web hacking techniques. SQL injection is the placement of malicious code in SQL statements, via web page input.	
HOSTING	If you want your web site to be able to store and retrieve data from a database, your web server should have access to a database-system that uses the SQL language. If your web server is hosted by an Internet Service Provider (ISP), you	

	will have to look for SQL hosting plans. The most common SQL hosting databases are MS SQL Server, Oracle, MySQL, and MS Access.	
DATA TYPES	The data type of a column defines what value the column can hold: integer, character, money, date and time, binary, and so on.	

W3SCHOOLS OPPGAVER


w3schools.com/sql/exercise.asp?filename=exercise_database7

Completed 52 of 52 Exercises:

SQL Functions ✓
SQL Like ✓
SQL Wildcards ✓
SQL In ✓
SQL Between ✓
SQL Alias ✓
SQL Join ✓
SQL Group By ✓

Exercise:

Congratulations!

You have finished all 52 SQL exercises.

Share your score:

Close

OPPGAVESETT 3

1. **Opprett tabell DEPT**
   (gjort i terminal)

2. **Opprett tabell EMP**
   (gjort I terminal)

3. **Legg til en verdi i DEPT**
   (gjort i terminal)

4. **List, etternavn, avdeling og lønn til alle ansatte med lønn mellom 1000 og 2000**

```
SELECT EMP.ENAME, EMP.SAL, DEPT.DNAME
FROM EMP
INNER JOIN DEPT ON EMP.DEPTNO = DEPT. DEPTNO
WHERE SAL BETWEEN 1000 AND 2000;
```

5. **List de ulike jobbtypene som finnes**

```
SELECT Distinct JOB
FROM EMP;
```

6. **List ansattnr, navn, jobb, lønn og avdelingsnr for ansatte i avdeling 10 og 30**

```
SELECT EMP, ENAME, JOB, SAL, DEPTNO
FROM EMP
WHERE DEPTNO = 10 AND DEPTNO = 30;
```

7. **Vis ansatte som ble rekruttert i 1982**

```
SELECT *
FROM EMP
```

```
WHERE HIREDATE =  “%82”
```

8. **List ansatte som har navn med TH eller AR i**

```
SELECT *
FROM EMP
WHERE ENAME LIKE ‘%TH%’ OR ENAME LIKE ‘%%AR’;
```

9. **List ansattnr og navn sortert på navn**

```
SELECT EMPNO, ENAME
FROM EMP
ORDER BY ENAME
```

10. **Finn navn, jobb, lønn ok kommisjon til alle ansatte som ikke har noen sjef**

```
SELECT ENAME, JOB, SAL, COMM
FROM EMP
WHERE MGR IS NULL
```

11. **List alle selgere i synkende rekkefølge på kommisjon delt på lønn**

```
SELECT *
FROM EMP
WHERE JOB IN (‘SALESMAN’)
SORT BY COMM/SAL DESC;
```

12. **Finn årlig kompensasjon til selgere basert på månedlig lønn og månedligkommisjon**

```
SELECT ENAME
(SAL*12) + (COMM*12) AS AARLIG
FROM EMP
WHERE JOB IN (‘SALESMAN’);
```

13. Finn alle selgere i avdeling 30 med lønn større eller lik 1500(pund)

```
SELECT *
FROM EMP
WHERE JOB ='SALESMAN' AND DEPTNO LIKE 30 AND SAL >= 1500;
```

14. Finn antall MANAGERS i ansatt tabellen

```
SELECT COUNT(*);
FROM EMP
WHERE JOB LIKE 'MANAGER';
```

15. Finn gjennomsnittelig årlig lønn lønn + kommisjon for selgerne

```
SELECT AVG(SAL + COMM)
FROM EMP
WHERE JOB = 'SALESMAN';
```

16. Finn differansen mellom høyeste lønn og laveste lønn

```
SELECT MAX(SAL)-MIN(SAL) AS DIFFERANSEN
FROM EMP
```

17. Finn det lengste avdelingsnavnet

```
SELECT DNAME,
MAX(LENGTH(DNAME)) AS LAvdelingsnavnet
FROM DEPT;
```

18. Finn antall mennesker i avdeling 30 som har fått kommisjon

```
SELECT COUNT(*)
FROM EMP
```

```
WHERE DEPTNO = ‘30’
WHERE COMM > 0;
```

19. Finn navn og lønn til alle ansatte i Chicago

```
SELECT ENAME, SAL, LOC
FROM EMP, DEP
WHERE LOC = ‘CHICAGO’;
```

20. List avdelingsnr, avdelingsnavn, jobb og etternavn med avdelingsnr i stigende rekkefølge

```
SELECT EMP, DEPTNO, DNAME, JOB, ENAME
FROM EMP, DEP
WHERE EMP,DEPTNO = DEPT.DEPTNO
ORDER BY DEPTNO;
```

21. List alle avdelinger som ikke har ansatt

```
SELECT DEPTNO
FROM DEPT
WHERE DEPTNO NOT IN (SELECT DEPTNO FROM EMP)

SELECT DNAME,
ENAME
FROM DEPT LEFT JOIN EMP ON
DEPT.DEPTNO = EMP.DEPNNO
WHERE ENAME IS NULL;
```

22. List alle avdelinger som har ansatte, og de avdelingene som ikke har noen ansatte.

```
SELECT DISTINCT DNAME
FROM DEPT LEFT JOIN EMP ON
```

```
DEPT.DEPTNO = EMP.DEPTNO;
```

23. Finn alle ansatte som tjener mer enn Jones

```
SELECT E1.ENAME, E1.SAL, E2.ENAME, E2.SAL
FROM EMP AS E1, EMP AS E2
WHERE E2.ENAME = ‘JONES’ AND E1.SAL>E2.SAL;
```

24. List ansatte som tjener mer enn sjefen sin

```
SELECT E1.*,E2*
FROM EMP AS E1, EMP AS E2
WHERE E1.MGR = E2.EMPNO AND E1.SAL > E2.SAL
```

25. List navn og jobb for ansatte som har samme jobb som Jones

```
SELECT E1.ENAME,
E1.JOB,
E2.ENAME,
E2.JOB
FROM EMP AS E1,
EMP AS E2
WHERE E2.ENAME = ‘JONES’
AND E1.JOB = E2.JOB;

SELECT ENAME, JOB
FROM EMP
WHERE JOB = (SELECT JOB FROM EMP WHERE ENAME = ‘JONES’)
AND ENAME <> ‘JONES’;
```

26. Finn alle ansatt i avdeling 10 som har samme jobb som noen i avdeling 30

```
SELECT E1.ENAME;
E1.JOB,
E1.DEPTNO,
E2.ENAME,
E2,DEPTNO
FROM EMP AS E1,
EMP AS E2
WHERE E1.DEEPTNO = 10
AND E2.DEPTNO = 30
AND E1.JOB = E2.JOB;
```

27. List navn og jobb til ansatte som har samme jobb og lik lønn som FORD

```
SELECT ENAME, JOB
FROM EMP
WHERE JOB = (SELECT JOB FROM EMP WHERE ENAME = ‘FORD’)
AND SAL = (SELECT SAL FROM EMP WHERE ENAME ‘FORD’) AND
ENAME <> ‘FORD’;

SELECT E1.ENAME, E1.JOB
FROM EMP AS E1, EMP AS E2
WHERE E2.ENAME = ‘FORD’
AND E1.JOB = E2.JOB
AND E1.SAL = E2.SAL;
```

28. List navn, jobb, avdeling og lønn for ansatte som har samme jobb som JONES og tørre eller lik lønn som FORD

```
SELECT E3.ENAME, E3.SAL, E3.JOB
FROM EMP AS E1, EMP AS E2, EMP AS E3
WHERE E1.ENAME = ‘JONES’ AND E3.JOB = E1.JOB OR E2.ENAME =
‘FORD’ AND E3.SAL > E2.SAL;

SELECT E1.ENAME, E1.JOB, E1.DEPTNO, E.SAL
FROM EMP AS E1, EMP AS E2, EMP AS E3
WHERE E2.ENAME = ‘JONES’
```

```
AND E3.ENAME = 'FORD'
AND E1.JOB = E2.JOB
AND E1.SAL >= E3.SAL;
```

29. Finn alle ansatte i avdeling 10 som har samme jobb som noen i SALES avdelingen

```
SELECT DISTINT E1.ENAME
FROM EMP AS E1,
EMP AS E2
WHERE E2.DEPTNO = 30
AND E1.JOB = E2.JOB;
```

30. Finn ansatte i Chicago som har samme jobb som ALLEN og sorter navnene i

```
        SELECT EMP.*
→FROM DEPT, EMP
→WHERE DEPT.DEPTNO = EMP.DEPTNO AND
→JOB = (SELECT JOB FROM EMP WHERE ENAME = 'ALLEN')
→AND LOC='CHICAGO'
→GROUP BY EMP.ENAME;
```

```
SELECT E1.ENAME
FROM EMP AS E1,
EMP AS E2
WHERE E2.ENAME = 'ALLEN'
AND E1.DEPTNO = 30
AND E1.JOB >= E2.JOB;
```

31. Finn alle ansatte som tjener mer enn gjennomsnittet for ansatte i sin avdeling

```
SELECT E1.ENAME, E1.SAL
FROM EMP AS E1
WHERE E1.SAL > (SELECT AVG(E2. SAL)
FROM EMP AS E2
WHERE E1.DEPTNO = E2.DEPTNO)
```

OPPGAVE 1 TIL KAP 4 I BOKA

a) Anta at vi har 161 varer plassert I 21 kategorier. Hvor mange rader gir spørringen SELECT * FROM Vare, Kategori?

Spørringen gir 21 * 161 = 3381 rader, ettersom alle rader fra Vare krysskobles med dem fra Kategori.

b) Hvor mange rader vil en likekobling av tabellene Vare og Kategori med hensyn på KatNr inneholde? Skriv SQL-koden. Hva skjer hvis noen av varene ikke er plassert i en kategori?

Kategori har nullmerker i kollonen KatNr dersom noen av varene ikke plasseres.

```
SELECT *
FROM VARE INNER JOIN Katergori
ON Vare.KatNr = Kategori.Kat.Nr;
```

c) Vis alle ordrelinje, men ta dessuten med varenavn (betegnele) og ordredato i utskriften:

```
SELECT ORDRELINJE.*,
VARE.BETEGNELSE,
ORDRE.ORDREDATO
FROM ORDRELINJE,
ORDRE,
VARE
WHERE ORDRELINJE.OrdreNr = Ordre.OrdreNr
AND Ordrelinje.VNr = Vare.VNr;
```

d) Utvid SQL-koden fra oppgave 1c med en kolone som viser totalbeløp for hver ordrelinje:

```
SELECT OL. *,
V.Betegnelse,
O.Ordredato,
OL.Antall * OL.Pris AS TotalBeløp
FROM Ordrelinje AS OL,
Ordre AS O,
Vare AS V
WHERE OL.OrdreNr = O.OrdreNr
AND OL.VNr = V.VNr
```

e) Vis samlet beløp hver kunde har handlet for:

```
SELECT Kunde.KNr, Kunde.Fornavn, Kunde.Etternavn,
SUM (Ordrelinje.Antall * Ordrelinje.Pris) AS Belop
FROM Ordrelinje, Ordre, Vare, kunde
WHERE Ordrelinje.OrdreNr = Ordre.OrdreNr
AND Ordrelinje.VNr = Vare.VareNr
AND Kunde.KNr = Ordre.KNr
GROU BY Kundre.KNr, Kunde.Fornavn, Kundre.Etternavn;
```

f) Prøv å utvide SQL-koden fra oppgave 1e med en ny kolonne som viser antall ordrer for hver kunde. Hva er problemet?

g) Vis samler beløp per ordre:

```
SELECT OrdreNr, SUM(Antall * PrisprEnhet) As
BelopPrOrdre
From Ordrelinje
GROUP BY OrdreNr;
```

h) Lag en vareliste som for hver vare viser antall enheter på lager og samlet lagerverdi for denne varen:

```
SELECT VNr, Antall, Antall * Pris AS LagerVerdi
FROM Vare;
```

i) Finn samlet verdi av varelageret:

```
SELECT SUM(Antall*Pris) AS VerdiAvLageret
FROM Vare;
```

j) Finn ut hvor mye hver varekategori har solgt for, Lag en sortert liste med bestselgerne først, og få navn på kategori i utskriften:

k) Vis alle postnumre der det enten bor en ansatt eller en kunde. Hvordan få med steder der det bor både en ansatt og en kunde? Hva kan du gjøre for å få med navn på poststedet?

```
SELECT DISTINCT Posted.PostNr
FROM Ansatt, Kunde, Poststed
WHERE Ansatt.PostNr = Posted.PostNr OR Kunde.PostNr =
Poststed.PostNr;
```

For å få med både navn og poststedet kan OR erstattes med AND.

Bibliografi

W3Schools. (1999-2022). *SQL Tutorials, SQL Database*. Hentet fra W3Schools: https://www.w3schools.com/sql/default.asp