STRUCTURED QUERY LANGUAGE
(SQL)

DATABASE MANAGEMENT SYSTEM (DBMS)

AND PROGRAMMING

DATA DICTIONARY

DATA DICTIONARY (METADATA)

- Data elements in a database or data model with detailed description of its format, relationships, meaning, source and usage.

EMPLOYEE

Ssn	Firstname	Lastname	City	Province	Postcode	Salary	Birthdate	Sex	Dno	Super_ssn

EMPLOYEE

Attribute name	Data type	Size	Key	Description
Ssn	int	20	Primary	Social security number of employee
Firstname	varchar	50		First name of employee
Lastname	varchar	50		Last name of employee
City	varchar	100		City of address of employee
Province	varchar	100		Province of address of employee
Postcode	varchar	100		Postcode of address of employee
Salary	float	9,2		Salary of employee
Birthdate	date			Birth date of employee
Sex	tinyint	1		Sex of employee
Dno	int	10	Foreign	Department number
Super_ssn	int	10	Foreign	Social security number of Supervisor

DATA TYPE IN MYSQL

String Data Types

Data type	Description
CHAR(size)	A FIXED length string (can contain letters, numbers, and special characters). The *size* parameter specifies the column length in characters - can be from 0 to 255. Default is 1
VARCHAR(size)	A VARIABLE length string (can contain letters, numbers, and special characters). The *size* parameter specifies the maximum column length in characters - can be from 0 to 65535
BINARY(size)	Equal to CHAR(), but stores binary byte strings. The *size* parameter specifies the column length in bytes. Default is 1
VARBINARY(size)	Equal to VARCHAR(), but stores binary byte strings. The *size* parameter specifies the maximum column length in bytes.
TINYBLOB	For BLOBs (Binary Large OBjects). Max length: 255 bytes
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT(size)	Holds a string with a maximum length of 65,535 bytes
BLOB(size)	For BLOBs (Binary Large OBjects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large OBjects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LONGBLOB	For BLOBs (Binary Large OBjects). Holds up to 4,294,967,295 bytes of data
ENUM(val1, val2, val3, ...)	A string object that can have only one value, chosen from a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. The values are sorted in the order you enter them
SET(val1, val2, val3, ...)	A string object that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values in a SET list

https://www.w3schools.com/mysql/mysql_datatypes.asp

DATA TYPE IN MYSQL

Number Data Types

Data type	Description
BIT(*size*)	A bit-value type. The number of bits per value is specified in *size*. The *size* parameter can hold a value from 1 to 64. The default value for *size* is 1.
TINYINT(*size*)	A very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255. The *size* parameter specifies the maximum display width (which is 255)
BOOL	Zero is considered as false, nonzero values are considered as true.
BOOLEAN	Equal to BOOL
SMALLINT(*size*)	A small integer. Signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The *size* parameter specifies the maximum display width (which is 255)
MEDIUMINT(*size*)	A medium integer. Signed range is from -8388608 to 8388607. Unsigned range is from 0 to 16777215. The *size* parameter specifies the maximum display width (which is 255)
INT(*size*)	A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The *size* parameter specifies the maximum display width (which is 255)
INTEGER(*size*)	Equal to INT(size)
BIGINT(*size*)	A large integer. Signed range is from -9223372036854775808 to 9223372036854775807. Unsigned range is from 0 to 18446744073709551615. The *size* parameter specifies the maximum display width (which is 255)
FLOAT(*size*, *d*)	A floating point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions
FLOAT(*p*)	A floating point number. MySQL uses the *p* value to determine whether to use FLOAT or DOUBLE for the resulting data type. If *p* is from 0 to 24, the data type becomes FLOAT(). If *p* is from 25 to 53, the data type becomes DOUBLE()
DOUBLE(*size*, *d*)	A normal-size floating point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter
DOUBLE PRECISION(*size*, *d*)	
DECIMAL(*size*, *d*)	An exact fixed-point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. The maximum number for *size* is 65. The maximum number for *d* is 30. The default value for *size* is 10. The default value for *d* is 0.
DEC(*size*, *d*)	Equal to DECIMAL(size,d)

DATA TYPE IN MYSQL

Date and Time Data Types

Data type	Description
DATE	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'
DATETIME(*fsp*)	A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time
TIMESTAMP(*fsp*)	A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP in the column definition
TIME(*fsp*)	A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'
YEAR	A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000. MySQL 8.0 does not support year in two-digit format.

BASIC SQL

DATABASE LANGUAGE (DBL)

- **Data Definition Language (DDL)** – A Language that define database conceptual schema.
  - CREATE
  - DROP
  - ALTER
- **Data Manipulation Language (DML)** – A Language that manipulate the data held in the database.
  - SELECT
  - INSERT
  - UPDATE
  - DELETE
- **Structured Query Language (SQL)** is a standard language for storing, manipulating and retrieving data in databases

DATA DEFINITION LANGUAGE (DDL)

DATABASE LANGUAGE (DBL)

- **Data Definition Language (DDL)** – A Language that define database conceptual schema.
  - CREATE
  - DROP
  - ALTER

CREATING A DATABASE

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

```
CREATE DATABASE databasename;
```

Example

```
CREATE DATABASE test2;
```

DELETING A DATABASE

- The DROP DATABASE statement is used to create a new MySQL database.

```
DROP DATABASE databasename;
```

Example

```
DROP DATABASE COMPANY;
```

Be careful before dropping a database. Deleting a database will result in loss of complete information stored in the database!

CREATING A TABLE

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

```
CREATE TABLE table_name (
    attribute1 datatype [NOT NULL] [DEFAULT 'default value'],
    attribute2 datatype [NOT NULL] [DEFAULT 'default value'],
    attribute3 datatype [NOT NULL] [DEFAULT 'default value'],
   ....
       [PRIMARY KEY (attribute),]
       [UNIQUE (attribute),]
       [FOREIGN KEY (attribute) REFERENCES parentTableName(attribute),]
       [CHECK (searchCondition)]
);
```

CREATING A TABLE

Example

```
CREATE TABLE Employee (
        Ssn int NOT NULL,
        FirstName varchar(50) NOT NULL DEFAULT ‘John’,
        LastName varchar(50),
        City varchar(100),
        Salary float(9,2) NOT NULL,
        Dno int,
        PRIMARY KEY (Ssn),
        UNIQUE (FirstName),
        FOREIGN KEY (Dno) REFERENCES Department(Dnumber),
        CHECK(Salary >= 1000)
);
```

DROP A TABLE

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

```
DROP TABLE table_name;
```

Example

```
DROP TABLE Employee;
```

ALTER A TABLE

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

```
ALTER TABLE table_name
ADD column_name datatype [FIRST|AFTER existing_column],
[ADD column_name datatype [FIRST|AFTER
existing_column]];
```

Example

```
ALTER TABLE Employee
ADD Sex tinyint(1) NOT NULL AFTER Lastname,
ADD Postcode char(6);
```

ALTER A TABLE

Change the data type

```
ALTER TABLE table_name
MODIFY COLUMN column_name datatype,
[MODIFY COLUMN column_name datatype];
```

Example

```
ALTER TABLE Employee
MODIFY COLUMN Ssn char(9);
```

ALTER A TABLE

Delete the column

```
ALTER TABLE table_name
DROP COLUMN column_name,
[DROP COLUMN column_name];
```

Example

```
ALTER TABLE Employee
DROP COLUMN Sex;
```

ALTER A TABLE

Change key

```
ALTER TABLE table_name
ADD|DROP FOREIGN KEY (key) REFERENCES reference_table_name(ref_key)
[ON UPDATE|DELETE NO ACTION|CASCADE];
```

Example

```
ALTER TABLE Employee
ADD FOREIGN KEY (Dno) REFERENCES Department(dnumber);
```

AUTO INCREMENT

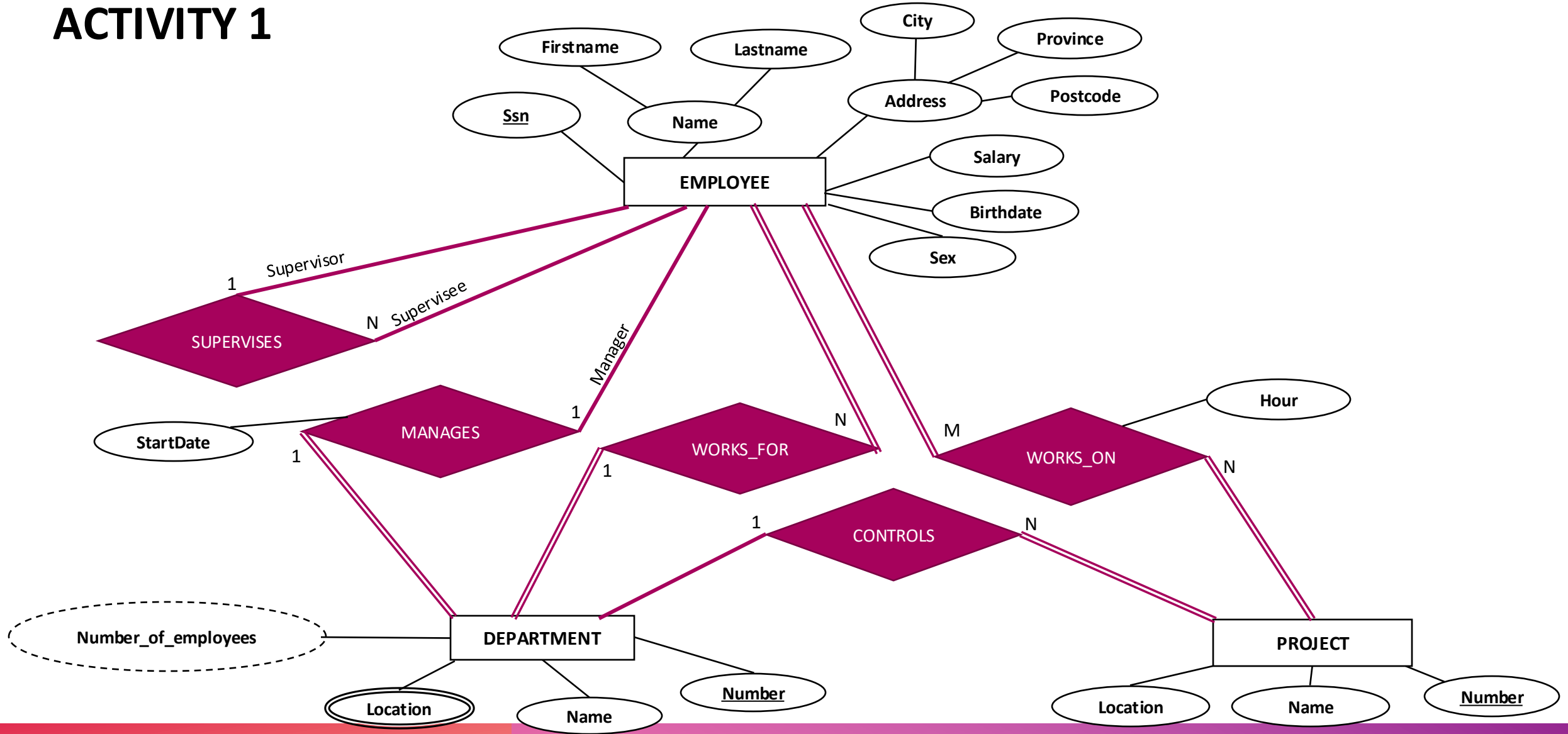
```
CREATE TABLE table_name (
    attribute1 datatype [NOT NULL] [DEFAULT ‘default value’] AUTO INCREMENT,
    attribute2 datatype [NOT NULL] [DEFAULT ‘default value’],
    attribute3 datatype [NOT NULL] [DEFAULT ‘default value’],
   ....
       [PRIMARY KEY (attribute),]
       [UNIQUE (attribute),]
       [FOREIGN KEY (attribute) REFERENCES parentTableName(attribute),]
       [CHECK (searchCondition)]
);
```

AUTO INCREMENT

Example

```
CREATE TABLE Employee (
      Ssn int NOT NULL AUTO_INCREMENT,
      FirstName varchar(50) NOT NULL DEFAULT ‘John’,
      LastName varchar(50),
      City varchar(100),
      Salary float(9,2) NOT NULL,
      Dno int,
      PRIMARY KEY (Ssn),
);
```

ACTIVITY 1


Firstname
Lastname
City
Province
Address
Postcode
Ssn
Name
EMPLOYEE
Salary
Birthdate
Sex
Supervisor
1
N
Supervisee
SUPERVISES
Manager
1
MANAGES
StartDate
1
N
WORKS_FOR
M
WORKS_ON
Hour
N
1
N
CONTROLS
Number_of_employees
DEPARTMENT
PROJECT
Location
Name
Number
Location
Name
Number

DATA DICTIONARY (METADATA)

EMPLOYEE

Attribute name	Data type	Size	Key	Description
Ssn	char	9	Primary	Social security number of employee
Firstname	varchar	50		First name of employee
Lastname	varchar	50		Last name of employee
City	varchar	100		City of address of employee
Province	varchar	100		Province of address of employee
Postcode	varchar	100		Postcode of address of employee
Salary	float	9,2		Salary of employee
Birthdate	date			Birth date of employee
Sex	tinyint	1		Sex of employee
Dno	int	6		Department number
Super_ssn	char	9		Social security number of Supervisor

MYSQL 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** - 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

TOOL

Sqlite

https://sqliteonline.com/

ACTIVITY 2

Using Sqlite online to crate table and insert data of question 1

DATA MANIPULATION LANGUAGE (DML)

DATABASE LANGUAGE (DBL)

- **Data Manipulation Language (DML)** – A Language that manipulate the data held in the database.
  - SELECT
  - INSERT
  - UPDATE
  - DELETE

SELECTING DATA

Select all the fields available in the table

```
SELECT *
FROM table_name;
```

Example

```
SELECT *
FROM Employee;
```

SELECTING DATA

Select data from a database

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

Example

```
SELECT Ssn, Firstname, Lastname
FROM Employee;
```

SELECTING DATA

WHERE clause used to extract only records that satisfy a specified condition

```
SELECT column1, column2, ...
FROM table_name
WHERE condition [AND/OR/NOT] condition;
```

Example

```
SELECT Ssn, Firstname, Lastname FROM Employee
WHERE Salary >= 30000;
```

```
SELECT Ssn, Firstname, Lastname FROM Employee
WHERE Salary >= 30000 AND Firstname <> 'Smith';
```

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
AND, OR, NOT	AND, OR, NOT

LIKE OPERATOR

There are two wildcards often used in conjunction with the LIKE operator:

The percent sign (%) represents zero, one, or multiple characters

The underscore sign (_) represents one, single character

```
SELECT column1, column2, ...
FROM table_name
WHERE column_name LIKE searchtext;
```

LIKE OPERATOR

LIKE Operator	Description
WHERE *column_name*LIKE 'a%'	Finds any values that start with "a"
WHERE *column_name* LIKE '%a'	Finds any values that end with "a"
WHERE *column_name* LIKE '%or%'	Finds any values that have "or" in any position
WHERE *column_name* LIKE '_r%'	Finds any values that have "r" in the second position
WHERE *column_name* LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE *column_name* LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE *column_name* LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

SELECT DISTINCT

Return only distinct (different) values

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

Example

```
SELECT DISTINCT Ssn, Firstname, Lastname
FROM Employee;
```

INSERTING DATA

Insert new records in a table. (Specify both the column names)

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

Example

```
INSERT INTO  Employee (Ssn, Firstname, Lastname, Sex, City)
VALUES (‘123456342’, ‘Jane’, ‘Doe’, 2, ‘Chiang Mai’);
```

INSERTING DATA

Insert new records in a table. (Adding values for all the columns)

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

Example

```
INSERT INTO  Employee
VALUES ('123456342', 'Jane', 'Doe', 'Mueang', 'Chiang Mai',
'50000', 45000, …);
```

** Order of the values must be the same order as the columns in the table.**

UPDATING DATA

Modify the existing records in a table

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Example

```
UPDATE  Employee
SET Firstname =  'Sam', Salary = 50000
WHERE Ssn = '123456789';
```

DELETE DATA

Delete existing records in a table

```
DELETE FROM table_name WHERE condition;
```

TRUNCATE TABLE

Delete all tuples (rows) from the table

```
TRUNCATE TABLE table_name;
```

Example

```
TRUNCATE TABLE Employee;
```

TABLE AND COLUMN ALIAS

- Improve the readability of the queries.
  - Alias for Columns
  - You can only use column aliases in GROUP BY, ORDER BY, or HAVING clauses.

```
SELECT SUM(SALARY) AS SSUM
FROM EMPLOYEE;
```

```
SELECT SUM(SALARY) AS SSUM, COUNT(*) AS ECOUNT
FROM EMPLOYEE;
```

```
SELECT CONCAT_WS(' ', lname, fname) AS `Full name`
FROM EMPLOYEE
ORDER BY `Full name`;
```

TABLE AND COLUMN ALIAS

- Improve the readability of the queries.
  - Alias for Tables

```
SELECT *
FROM EMPLOYEE e;
```

```
SELECT e.fname, e.lname
FROM EMPLOYEE e
ORDER BY e.fname;
```

```sql
SELECT
    firstname AS fname,
    lastname  AS lname,
    firstname || " " || lastname AS fullname
FROM customers;
```

```sql
SELECT
    firstname AS fname,
    lastname  AS lname,
    firstname || " " || lastname AS fullname,
    firstname || "@google.com"   AS email
FROM customers;
```

```sql
SELECT
    firstname,
    country,
    company,
    email
FROM customers
WHERE LOWER(country) = 'france';
```

```sql
SELECT
    firstname,
    country,
    company,
    email
FROM customers
WHERE UPPER(country) = 'FRANCE' OR UPPER(country) = 'BRAZIL'
```

BETWEEN OPERATOR

The BETWEEN operator selects values within a given range.

```
SELECT column1, column2, ...
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

Example

```
SELECT *
FROM EMPLOYEE
WHERE Salary BETWEEN 30000 AND 70000;
```

IN OPERATOR

The IN operator allows you to specify multiple values in a WHERE clause.

```
SELECT column1, column2, ...
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

Example

```
SELECT *
FROM EMPLOYEE
WHERE Salary IN('John', 'James', 'Jennifer', 'Jill');
```

DATE

- **DATE**
  - DAY(date_column)
  - MONTH(date_column)
  - YEAR(date_column)

- Example

```
SELECT *
FROM EMPLOYEE
WHERE YEAR(bdate) = 1966;
```

ACTIVITY 3

Use a given database to query the customer information which are first name, last name, country and email of whom stay in USA by Sqlite online

ACTIVITY 4

Use a given database to query the customer information which are first name, last name, country and email of whom stay in USA, France and Brazil by Sqlite online

ACTIVITY 5

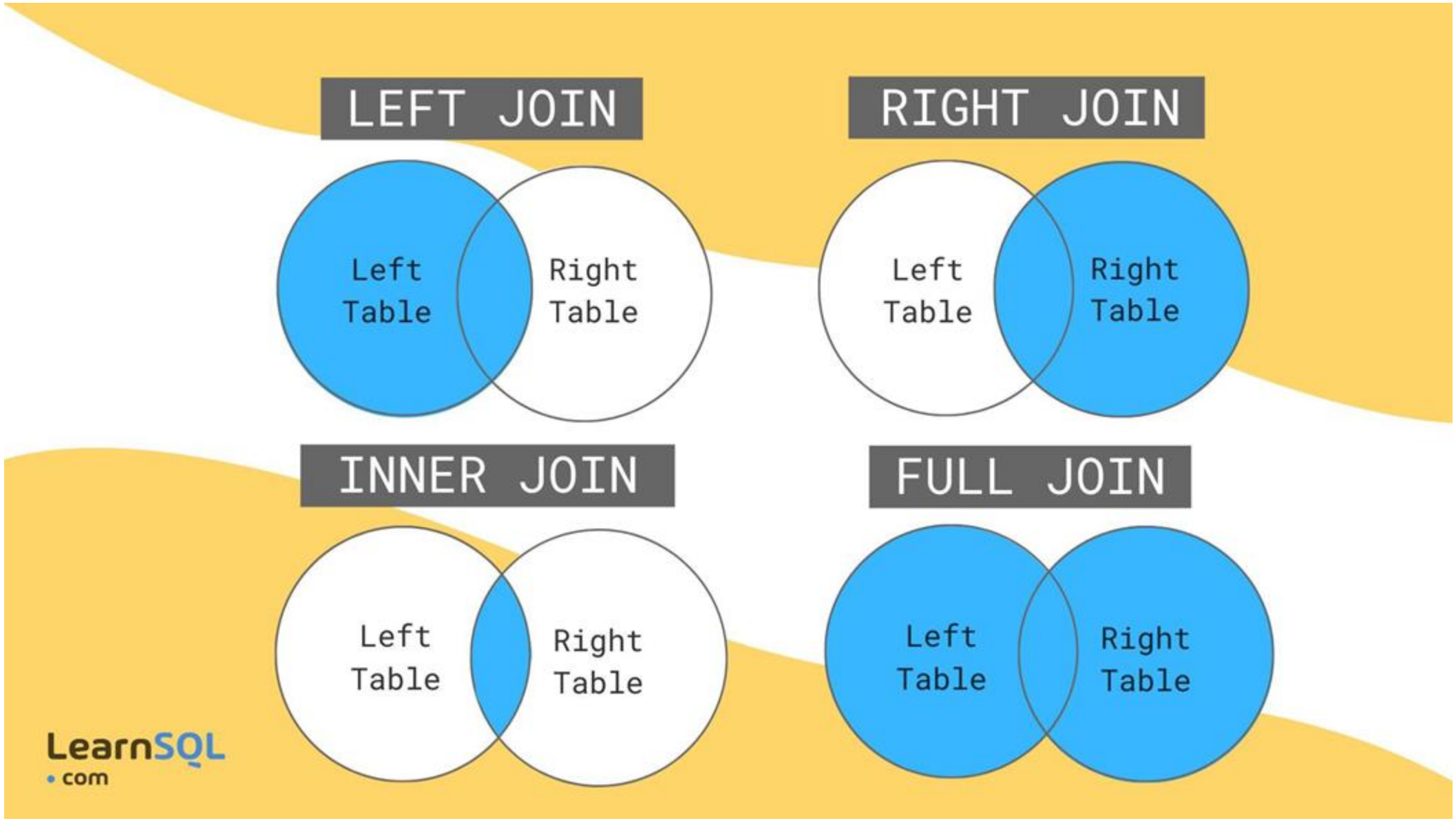
Use a given database to query the customer information which are first name, last name and email who use the email with domain “@yahoo”

ADVANCED SQL

JOIN

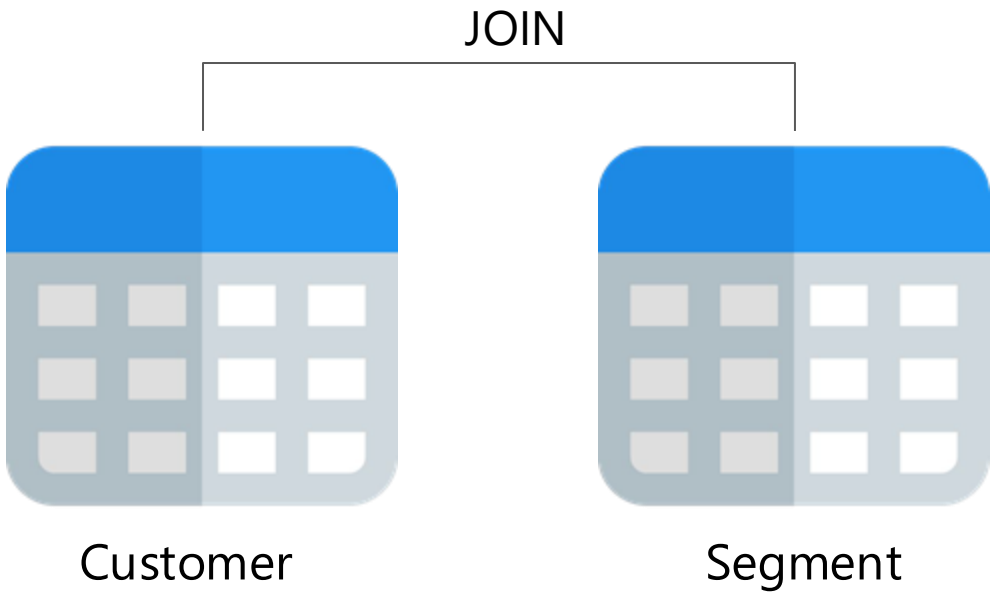
- A method of linking data between one or more tables based on values of the common column between the tables.

- MySQL supports the following types of joins
  - INNER JOIN (JOIN)
  - LEFT JOIN
  - RIGHT JOIN
  - CROSS JOIN


LEFT JOIN
Left Table
Right Table
RIGHT JOIN
Left Table
Right Table
INNER JOIN
Left Table
Right Table
FULL JOIN
Left Table
Right Table
LearnSQL.com

WHAT IS JOIN?

Getting data from multiple tables



Customer

ID	Name	City
1001	Toy	BKK
1002	Anna	LON
1003	Marry	LON

Segment

ID	SegName	Cust_ID
1	Deal Hunter	1001
2	Price Sensitive	1002
3	Premium	1003

Join PK=FK

RESULT

ID	Name	City	SegName
1001	Toy	BKK	Deal Hunter
1002	Anna	LON	Price Sensitive
1003	Marry	LON	Premium

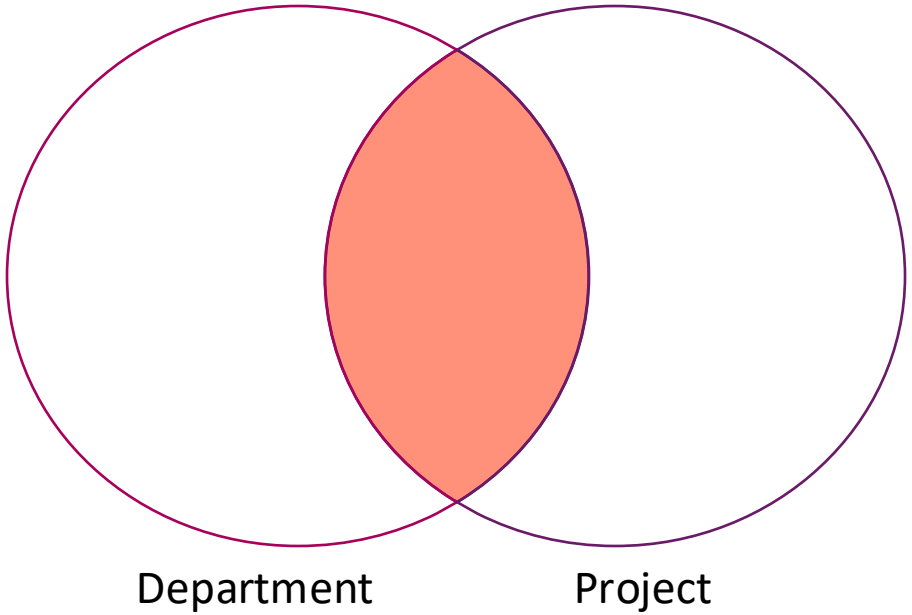
JOIN SYNTAX

```
SELECT columns
FROM table1
JOIN table2
ON join_condition;
```

```
SELECT columns
FROM table1
JOIN table2
ON table1.pk = table2.fk;
```

INNER JOIN

- The inner join clause compares each row from the first table with every row from the second table.
- The inner join clause includes only matching rows from both tables



INNER JOIN=JOIN(DEFAULT)

Customer

ID	Name	City
1001	Toy.	BKK
1002	Anna	LON
1003	Marry	LON
1004	Ken	JPN

Segment

ID	SegName	Cust_ID
1	Deal Hunter	1001
2	Price Sensitive	1002
3	Premium	1003

RESULT

Only Matched Rows Return

ID	Name	City	ID	SegName	Cust_ID
1001	Toy	BKK	1	Deal Hunter	1001
1002	Anna	LON	2	Price Sensitive	1002
1003	Marry	LON	3	Premium	1003

INNER JOIN OR JOIN

```
SELECT columns
FROM table1
INNER JOIN table2
ON join_condition;
```

```
SELECT columns
FROM table1
JOIN table2
ON join_condition;
```

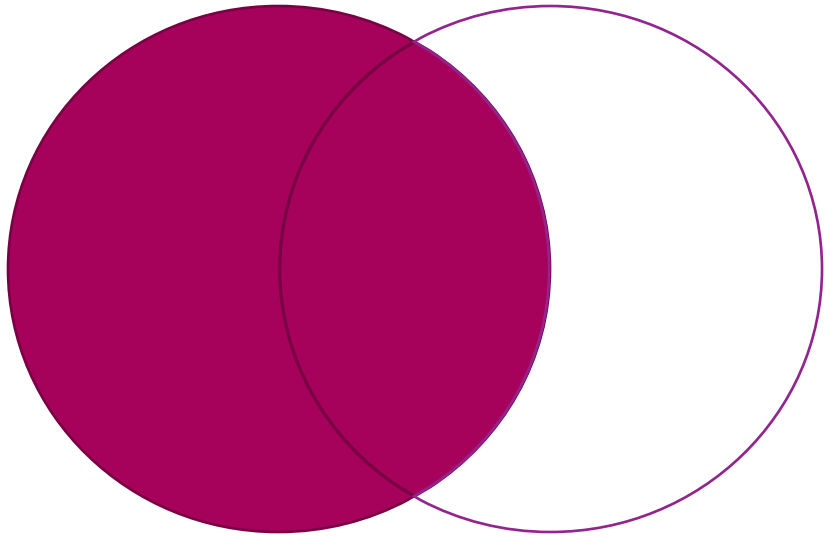
Example

```
SELECT *
FROM DEPARTMENT
INNER JOIN PROJECT
ON dnum = dnumber;
```

```
SELECT *
FROM DEPARTMENT as d
INNER JOIN PROJECT as p
ON p.dnum = d.dnumber;
```

LEFT JOIN

- The left join selects data starting from the left table. For each row in the left table, the left join compares with every row in the right table.
- The left join selects all data from the left table whether there are matching rows exist in the right table or not.



LEFT JOIN

Customer

ID	Name	City	
1001	Toy	BKK	
1002	Anna	LON	
1003	Marry	LON	
1004	Ken	JPN	

Segment

ID	SegName	Cust_ID
1	Deal Hunter	1001
2	Price Sensitive	1002
3	Premium	1003

RESULT

All rows in left table will be in the result set

ID	Name	City	ID	SegName	Cust_ID
1001	Toy	BKK	1	Deal Hunter	1001
1002	Anna	LON	2	Price Sensitive	1002
1003	Marry	LON	3	Premium	1003
1004	Ken	JPN	NULL	NULL	NULL

LEFT JOIN

```
SELECT columns
FROM table1
LEFT JOIN table2
ON join_condition;
```

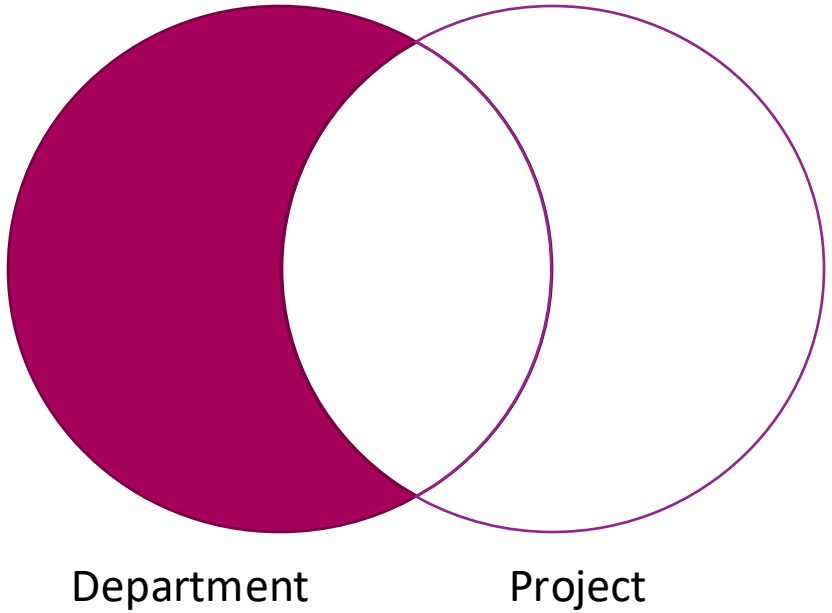
Example

```
SELECT *
FROM DEPARTMENT
LEFT JOIN PROJECT
ON dnum = dnumber;
```

```
SELECT *
FROM DEPARTMENT as d
LEFT JOIN PROJECT as p
ON p.dnum = d.dnumber;
```

LEFT JOIN

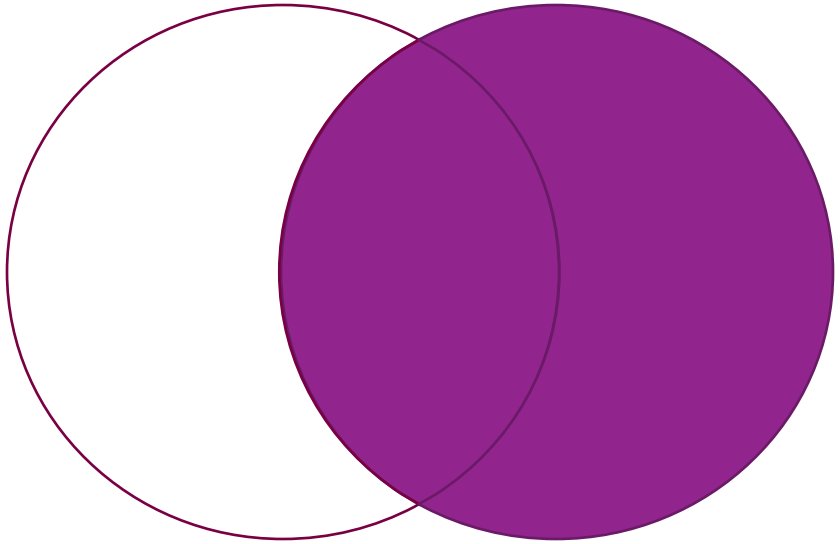
The left join with select rows that only exist in the left table




```
SELECT *
FROM DEPARTMENT d
LEFT JOIN PROJECT p
ON p.dnum = d.dnumber
WHERE p.dnum IS NULL;
```

RIGHT JOIN

- The right join selects data starting from the right table. For each row in the right table, the left join compares with every row in the left table.
- The right join selects all data from the right table whether there are matching rows exist in the left table or not.



Department Project

RIGHT JOIN

```
SELECT columns
FROM table1
RIGHT JOIN table2
ON join_condition;
```

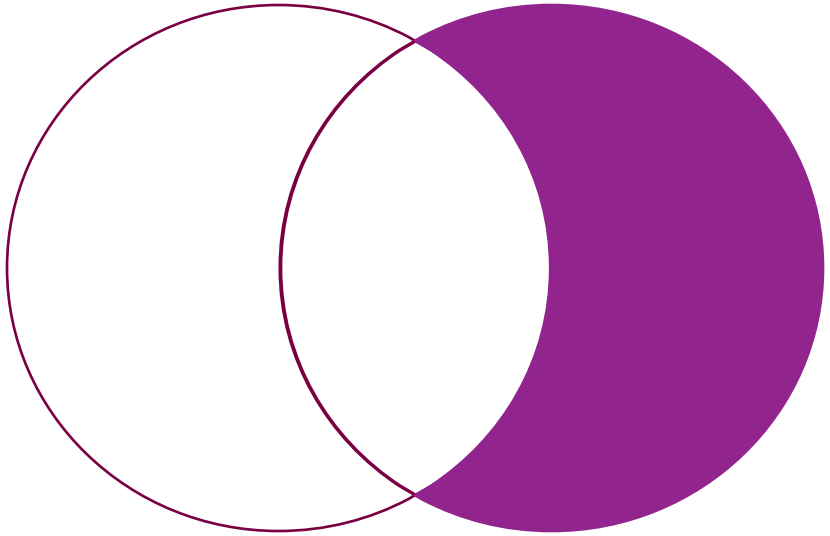
Example

```
SELECT *
FROM DEPARTMENT
RIGHT JOIN PROJECT
ON dnum = dnumber;
```

```
SELECT *
FROM DEPARTMENT as d
RIGHT JOIN PROJECT as p
ON p.dnum = d.dnumber;
```

RIGHT JOIN

The left join with select rows that only exist in the left table

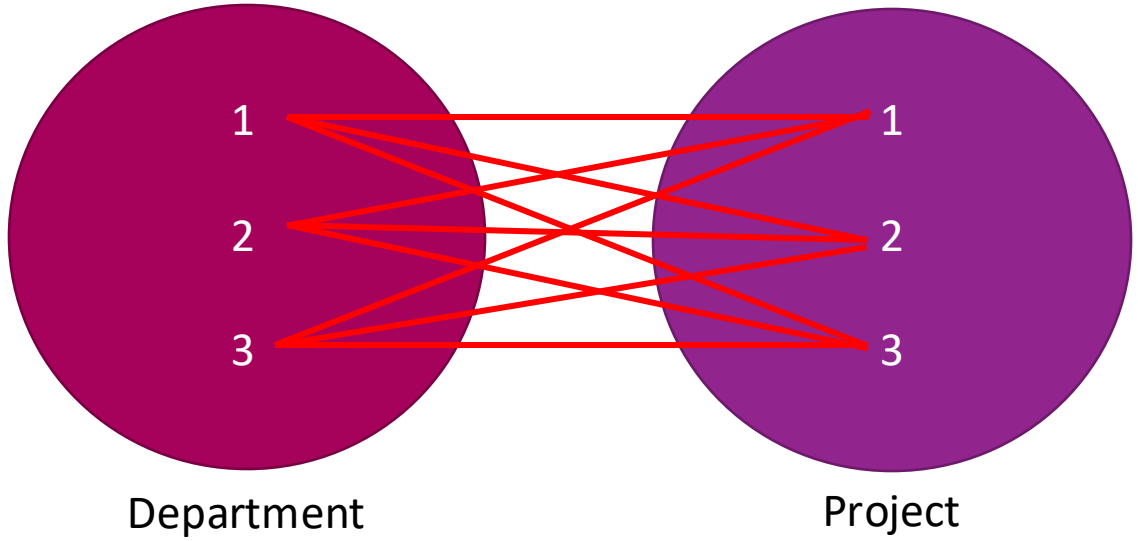



```
SELECT *
FROM DEPARTMENT d
RIGHT JOIN PROJECT p
ON p.dnum = d.dnumber
WHERE d.dnumber IS NULL;
```

CROSS JOIN

- The cross join makes a Cartesian product of rows from the joined tables.
- The cross join combines each row from the first table with every row from the right table to make the result set.
- the cross join clause does not have a join condition.
- Number of rows MxN

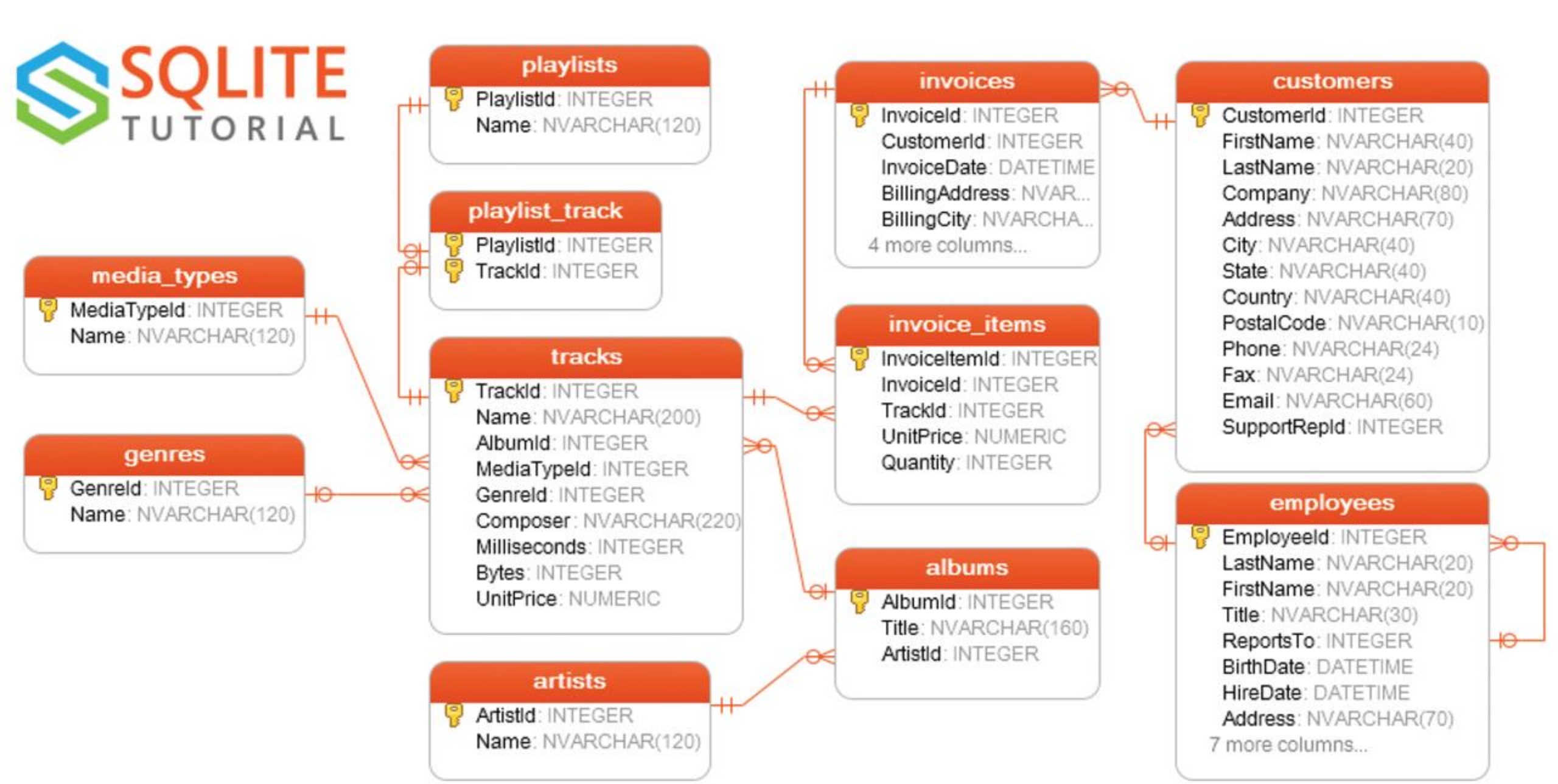
```
SELECT *
FROM PROJECT
CROSS JOIN DEPARTMENT;
```



SELF JOIN

- The self join is often used to query hierarchical data or to compare a row with other rows within the same table.
- To perform a self join, you must use table aliases to not repeat the same table name twice in a single query.

```
SELECT
       CONCAT(e.fname, ' ', e.lname) AS 'employee supervisor',
       CONCAT(s.fname, ' ', s.lname) AS employee
FROM EMPLOYEE s
LEFT JOIN EMPLOYEE e
ON e.ssn = s.superssn
ORDER BY 'employee supervisor';
```


SQLITE
TUTORIAL
playlists
PlaylistId: INTEGER
Name: NVARCHAR(120)
playlist_track
PlaylistId: INTEGER
TrackId: INTEGER
media_types
MediaTypeId: INTEGER
Name: NVARCHAR(120)
genres
GenreId: INTEGER
Name: NVARCHAR(120)
tracks
TrackId: INTEGER
Name: NVARCHAR(200)
AlbumId: INTEGER
MediaTypeId: INTEGER
GenreId: INTEGER
Composer: NVARCHAR(220)
Milliseconds: INTEGER
Bytes: INTEGER
UnitPrice: NUMERIC
artists
ArtistId: INTEGER
Name: NVARCHAR(120)
invoices
InvoiceId: INTEGER
CustomerId: INTEGER
InvoiceDate: DATETIME
BillingAddress: NVAR...
BillingCity: NVARCHA...
4 more columns...
invoice_items
InvoiceItemId: INTEGER
InvoiceId: INTEGER
TrackId: INTEGER
UnitPrice: NUMERIC
Quantity: INTEGER
albums
AlbumId: INTEGER
Title: NVARCHAR(160)
ArtistId: INTEGER
customers
CustomerId: INTEGER
FirstName: NVARCHAR(40)
LastName: NVARCHAR(20)
Company: NVARCHAR(80)
Address: NVARCHAR(70)
City: NVARCHAR(40)
State: NVARCHAR(40)
Country: NVARCHAR(40)
PostalCode: NVARCHAR(10)
Phone: NVARCHAR(24)
Fax: NVARCHAR(24)
Email: NVARCHAR(60)
SupportRepId: INTEGER
employees
EmployeeId: INTEGER
LastName: NVARCHAR(20)
FirstName: NVARCHAR(20)
Title: NVARCHAR(30)
ReportsTo: INTEGER
BirthDate: DATETIME
HireDate: DATETIME
Address: NVARCHAR(70)
7 more columns...

ACTIVITY 6

Use a given database to query the album information of each artist

ACTIVITY 7

Continue from question 6, join the tracks table to query the following information

- Artist name
- Album title
- Track name
- Track milliseconds

Special activity

Write an SQL query to find the **average salary** of employees who are working on projects in each department, and list this information along with the department name. Only include departments where the **average salary is greater than $50,000**.

```
SELECT d.DepartmentName, AVG(e.Salary) AS AverageSalary
FROM Departments d
JOIN Projects p ON d.DepartmentID = p.DepartmentID
JOIN EmployeeProjects ep ON p.ProjectID = ep.ProjectID
JOIN Employees e ON ep.EmployeeID = e.EmployeeID
GROUP BY d.DepartmentName
HAVING AVG(e.Salary) > 50000;
```

Special activity

Write an SQL query to retrieve the following details:

- The name of the artist
- The title of the album
- The name of the track
- The length of the track (in milliseconds)

The query should only include tracks with a length longer than 5 minutes (300,000 milliseconds). The results should be sorted by the artist's name, album title, and track length in descending order.

```
SELECT
    artists.Name,
    albums.Title,
    tracks.Name,
    tracks.Milliseconds
FROM
    tracks
JOIN
    albums ON tracks.AlbumId = albums.AlbumId
JOIN
    artists ON albums.ArtistId = artists.ArtistId
WHERE
    tracks.Milliseconds > 300000
ORDER BY
    artists.Name ASC,
    albums.Title ASC,
    tracks.Milliseconds DESC;
```