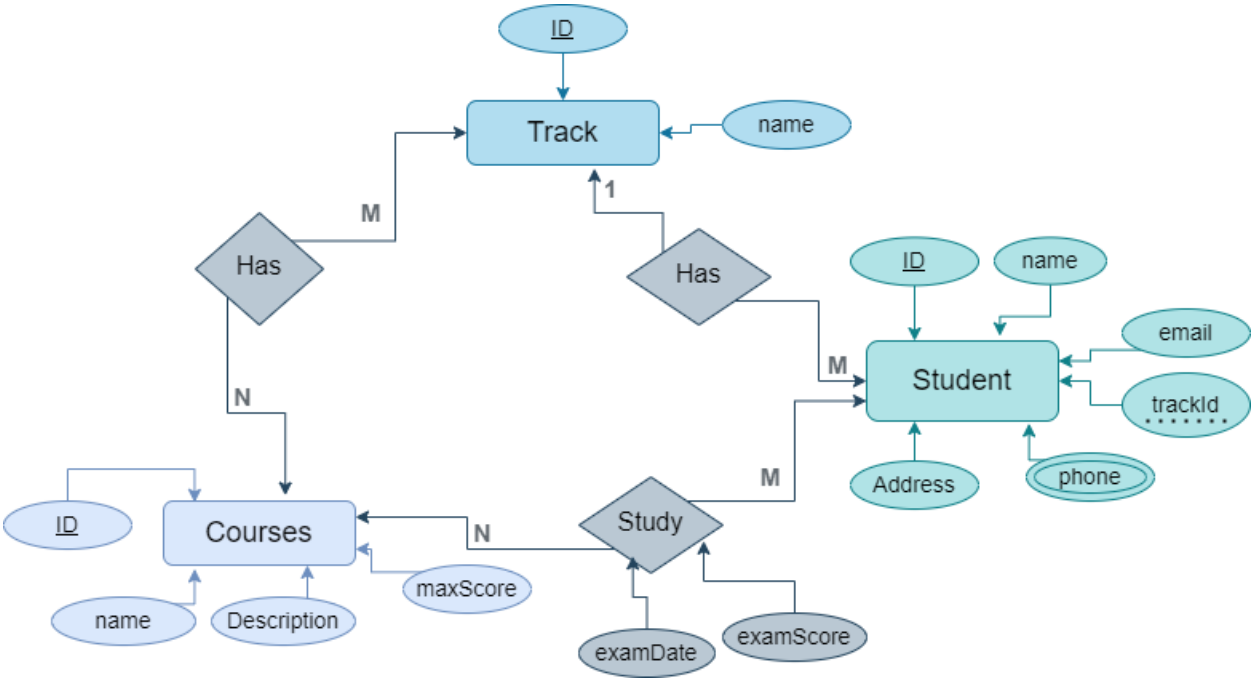
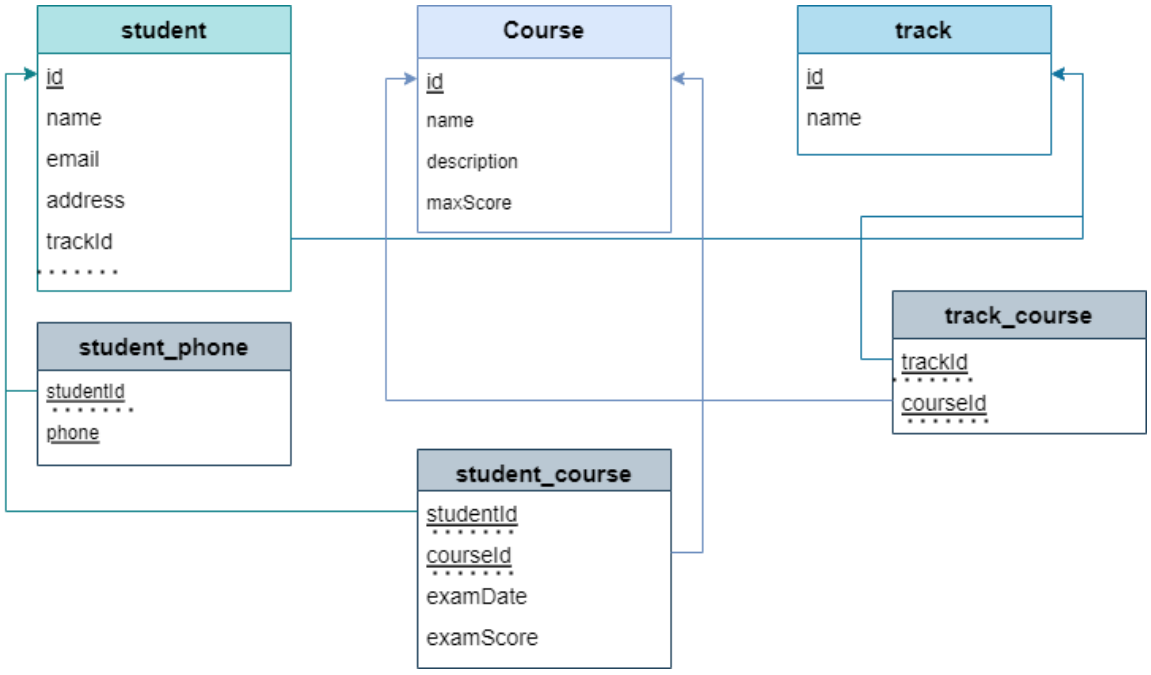
Lab 1

ERD Diagram



Mapping



Commands

- **CREATE DATABASE** iti;
- \c iti
- **CREATE TABLE** track
  (id int PRIMARY KEY, name text);
- **CREATE TABLE** student
  (id int PRIMARY KEY, name text, email text, address text, trackId int REFERENCES track (id));
- **CREATE TABLE** course
  (id int PRIMARY KEY, name text, description text, maxScore int);
- **CREATE TABLE** student_phone
  (studentId int REFERENCES student (id), phone int, PRIMARY KEY (studentId, phone));
- **CREATE TABLE** student_course
  (studentId int REFERENCES student (id), courseId int REFERENCES course (id), examDate date, examScore int, PRIMARY KEY (studentId, courseId));
- **CREATE TABLE** track_course
  (trackId int REFERENCES track (id), courseId int REFERENCES course (id), PRIMARY KEY (trackId, courseId));

```
INSERT INTO track VALUES
(1,'Full Stack Web Development Using Python'),
(2,'UI UX Design'),
(3,'Software Testing'),
(4,'2D Graphics Designer'),
(5,'Mobile Application');
INSERT INTO student VALUES
(1,'Sarah Mohamed','sarah@gmail.com','Minyat Elnasr',1),
(2,'Mryam Mahmoud','mryam@gmail.com','Elhamidya',4),
(3,'Eman Moaz','eman@gmail.com','Riyadh',5),
(4,'Omnia AbdelAziz','omnia@gmail.com','Russia',3),
(5,'Amira Abdallah','amira@gmail.com','El-Mizanya',2);
INSERT INTO course VALUES
(1,'HTML & CSS','Client Side Technologies',100),
(2,'Java Script','Client Side Technologies',100),
(3,'Python','Backend',100),
(4,'Database','Database Course',100),
(5,'Communication SKills','Soft Skill',100);
INSERT INTO student_phone VALUES (1,01065398515),
(1,0106295396),
(2,01065325863),
(3,01125678063),
(4,0129864329),
(5,01287644526);
```

- INSERT INTO student_course VALUES (1,1,'1/8/2023',100),
  (2,2,'12/7/2023',97),
  (3,5,'8/8/2023',95),
  (4,3,'30/7/2023',97),
  (5,4,'20/8/2023',90);
- INSERT INTO track_course VALUES
  (1,1),
  (4,2),
  (5,5),
  (3,3),
  (2,4);

SQL Shell (psql)

```
iti=# \d track
                Table "public.track"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | integer |           | not null |
 name   | text    |           |          |
Indexes:
    "track_pkey" PRIMARY KEY, btree (id)
Referenced by:
    TABLE "student" CONSTRAINT "student_trackid_fkey" FOREIGN KEY (trackid) REFERENCES track(id)
    TABLE "track_course" CONSTRAINT "track_course_trackid_fkey" FOREIGN KEY (trackid) REFERENCES track(id)


iti=# \d student
                Table "public.student"
 Column  |  Type   | Collation | Nullable | Default
---------+---------+-----------+----------+---------
 id      | integer |           | not null |
 name    | text    |           |          |
 email   | text    |           |          |
 address | text    |           |          |
 trackid | integer |           |          |
Indexes:
    "student_pkey" PRIMARY KEY, btree (id)
Foreign-key constraints:
    "student_trackid_fkey" FOREIGN KEY (trackid) REFERENCES track(id)
Referenced by:
    TABLE "student_course" CONSTRAINT "student_course_studentid_fkey" FOREIGN KEY (studentid) REFERENCES student(id)
    TABLE "student_phone" CONSTRAINT "student_phone_studentid_fkey" FOREIGN KEY (studentid) REFERENCES student(id)


iti=# \d course
                 Table "public.course"
   Column    |  Type   | Collation | Nullable | Default
-------------+---------+-----------+----------+---------
 id          | integer |           | not null |
 name        | text    |           |          |
 description | text    |           |          |
 maxscore    | text    |           |          |
Indexes:
```

SQL Shell (psql)

```
iti=# \d course
                 Table "public.course"
   Column    |  Type   | Collation | Nullable | Default
-------------+---------+-----------+----------+---------
 id          | integer |           | not null |
 name        | text    |           |          |
 description | text    |           |          |
 maxscore    | text    |           |          |
Indexes:
    "course_pkey" PRIMARY KEY, btree (id)
Referenced by:
    TABLE "student_course" CONSTRAINT "student_course_courseid_fkey" FOREIGN KEY (courseid) REFERENCES course(id)
    TABLE "track_course" CONSTRAINT "track_course_courseid_fkey" FOREIGN KEY (courseid) REFERENCES course(id)


iti=# \d student_phone
              Table "public.student_phone"
  Column   |  Type   | Collation | Nullable | Default
-----------+---------+-----------+----------+---------
 studentid | integer |           | not null |
 phone     | integer |           | not null |
Indexes:
    "student_phone_pkey" PRIMARY KEY, btree (studentid, phone)
Foreign-key constraints:
    "student_phone_studentid_fkey" FOREIGN KEY (studentid) REFERENCES student(id)


iti=# \d student_course
            Table "public.student_course"
  Column   |  Type   | Collation | Nullable | Default
-----------+---------+-----------+----------+---------
 studentid | integer |           | not null |
 courseid  | integer |           | not null |
 examdate  | date    |           |          |
 examscore | integer |           |          |
Indexes:
    "student_course_pkey" PRIMARY KEY, btree (studentid, courseid)
Foreign-key constraints:
    "student_course_courseid_fkey" FOREIGN KEY (courseid) REFERENCES course(id)
    "student_course_studentid_fkey" FOREIGN KEY (studentid) REFERENCES student(id)
```

```
iti=# \d track_course
             Table "public.track_course"
  Column  |  Type   | Collation | Nullable | Default
----------+---------+-----------+----------+---------
 trackid  | integer |           | not null |
 courseid | integer |           | not null |
Indexes:
    "track_course_pkey" PRIMARY KEY, btree (trackid, courseid)
Foreign-key constraints:
    "track_course_courseid_fkey" FOREIGN KEY (courseid) REFERENCES course(id)
    "track_course_trackid_fkey" FOREIGN KEY (trackid) REFERENCES track(id)
```

SQL Shell (psql)

```
iti=# SELECT * FROM track;
 id |                  name
----+----------------------------------------
  1 | Full Stack Web Development Using Python
  2 | UI UX Design
  3 | Software Testing
  4 | 2D Graphics Designer
  5 | Mobile Application
(5 rows)


iti=# SELECT * FROM student;
 id |      name       |      email      |    address    | trackid
----+-----------------+-----------------+---------------+---------
  1 | Sarah Mohamed   | sarah@gmail.com | Minyat Elnasr |       1
  2 | Mryam Mahmoud   | mryam@gmail.com | Elhamidya     |       4
  3 | Eman Moaz       | eman@gmail.com  | Riyadh        |       5
  4 | Omnia AbdelAziz | omnia@gmail.com | Russia        |       3
  5 | Amira Abdallah  | amira@gmail.com | El-Mizanya    |       2
(5 rows)


iti=# SELECT * FROM course;
 id |         name          |       description        | maxscore
----+-----------------------+--------------------------+----------
  1 | HTML & CSS            | Client Side Technologies |      100
  2 | Java Script           | Client Side Technologies |      100
  3 | Python                | Backend                  |      100
  4 | Database              | Database Course          |      100
  5 | Communication SKills | Soft Skill               |      100
(5 rows)


iti=# SELECT * FROM student_phone;
 studentid |   phone
-----------+------------
         1 | 1065398515
         1 |  106295396
         2 | 1065325863
         3 | 1125678063
```


SQL Shell (psql)

```
iti=# SELECT * FROM student_phone;
 studentid |   phone
-----------+------------
         1 | 1065398515
         1 |  106295396
         2 | 1065325863
         3 | 1125678063
         4 |  129864329
         5 | 1287644526
(6 rows)


iti=# SELECT * FROM student_course;
 studentid | courseid |  examdate  | examscore
-----------+----------+------------+-----------
         1 |        1 | 2023-08-01 |       100
         2 |        2 | 2023-07-12 |        97
         3 |        5 | 2023-08-08 |        95
         4 |        3 | 2023-07-30 |        97
         5 |        4 | 2023-08-20 |        90
(5 rows)


iti=# SELECT * FROM track_course;
 trackid | courseid
---------+----------
       1 |        1
       4 |        2
       5 |        5
       3 |        3
       2 |        4
(5 rows)


iti=#
```

Reports

What is NoSQL?

NoSQL, which stands for "not only SQL," is a term used to describe a category of database management systems (DBMS) that differ from traditional relational databases. NoSQL databases are designed to handle large-scale distributed data storage and processing requirements, particularly for unstructured and semi-structured data.

Here are some key characteristics and features of NoSQL databases:

Schema-less: NoSQL databases are typically schema-less, meaning that they do not require a predefined fixed schema. Data can be stored in various formats, such as key-value pairs, documents, graphs, or wide-column stores, allowing for flexibility in handling diverse data types.

Horizontal scalability: NoSQL databases are designed to scale horizontally, which means they can handle increasing amounts of data and traffic by adding more commodity servers to the database cluster. This makes them suitable for big data applications with high scalability requirements.

High performance: NoSQL databases often prioritize performance and scalability over complex relational operations. They are optimized for handling large volumes of data and high-speed data ingestion and retrieval.

Distributed architecture: NoSQL databases are built with distributed architectures that allow data to be spread across multiple servers or nodes in a cluster. This enables data replication, fault tolerance, and high availability.

Flexible data models: Different NoSQL databases use various data models, including key-value, document, column-family, and graph models. This flexibility allows developers to choose the most appropriate data model for their specific application requirements.

Designed for specific use cases: NoSQL databases are often designed to address specific use cases, such as real-time analytics, content management, caching, social networking, or IoT data management. Each type of NoSQL database has its own strengths and trade-offs.

It's important to note that while NoSQL databases offer advantages in scalability and flexibility, they may not be suitable for all types of applications. The choice between NoSQL and traditional SQL databases depends on the specific requirements, data model, consistency needs, and scalability demands of the application at hand.

~~~~~~

DBMSs Types (such as: Hierarchical,…) and brief about each type?

There are several types of database management systems (DBMS) based on their data models and structures. Here are some of the commonly known types:

1. Hierarchical DBMS: In a hierarchical database model, data is organized in a tree-like structure, where each record has a parent-child relationship. Each parent can have multiple children, but a child can have only one parent. This type of DBMS is suitable for representing one-to-many relationships. However, hierarchical DBMSs can become complex and
~~~~~~

inflexible when dealing with data that doesn't fit well into a hierarchical structure.

2. Network DBMS: The network database model is similar to the hierarchical model but allows more flexible relationships between records. In a network DBMS, data is organized using sets and links. Records can have multiple parent and child records, and relationships between records are established through these links. Network DBMSs can handle complex relationships more effectively than hierarchical DBMSs, but they can be difficult to design and maintain.

3. Relational DBMS: Relational databases are based on the relational model, which organizes data into tables with rows and columns. Data is stored in related tables, and relationships between tables are established through keys. The relational model provides a structured and standardized approach to data storage and retrieval, with support for complex queries, integrity constraints, and transaction management. Relational DBMSs, such as MySQL, Oracle, and PostgreSQL, are widely used in various applications.

4. Object-Oriented DBMS (OODBMS): Object-oriented databases store data in the form of objects, which encapsulate both data and behavior. They allow the storage of complex data types and support inheritance, polymorphism, and encapsulation. OODBMSs are well-suited for object-oriented programming languages and applications that require the persistence of complex object structures. Examples of OODBMSs include ObjectDB and db4o.

5. Document DBMS: Document databases, also known as document-oriented DBMSs, store and retrieve data in the form of semi-structured documents, typically in JSON or XML formats. Document DBMSs provide flexibility in data representation and schema evolution, allowing each document to have its own structure. They are commonly used for content management, blogging platforms, and applications that deal with unstructured or rapidly evolving data. MongoDB and CouchDB are popular document DBMSs.

6. Columnar DBMS: Columnar databases store data in columns rather than rows. This column-oriented storage allows for efficient data compression, faster query performance, and selective column retrieval. Columnar DBMSs are often used for analytical workloads that involve aggregations and reporting. Examples include Apache Cassandra, Apache HBase, and Google Bigtable.

7. Graph DBMS: Graph databases are designed to represent and store data as nodes, edges, and properties. They excel at managing highly connected data and relationships between entities. Graph DBMSs are commonly used in applications such as social networks, recommendation engines, and network analysis. Examples include Neo4j, Amazon Neptune, and JanusGraph.

Each type of DBMS has its own strengths and weaknesses, and the choice depends on the specific requirements, data model, and use case of the application.