



Database

Web Application

Suriya Natsupakpong, PhD

Institute of Field Robotics (FIBO)

King Mongkut's University of Technology Thonburi (KMUTT)

Database Management System (DBMS)

- The Database Management System (DBMS) is software provided by the database vendor.
- Software products such as Microsoft Access, Oracle, Microsoft SQL Server, Sybase, DB2, INGRES, and MySQL are all DBMSs.

ACID

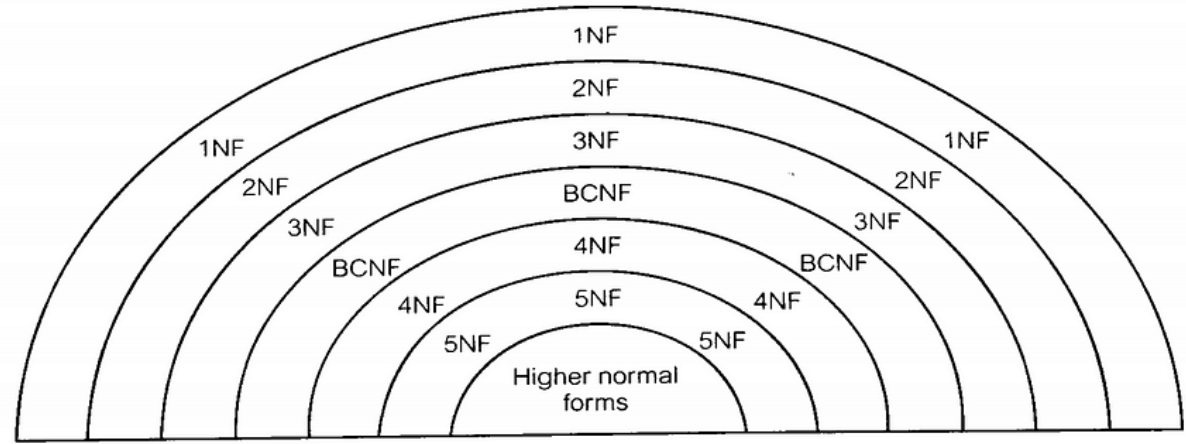
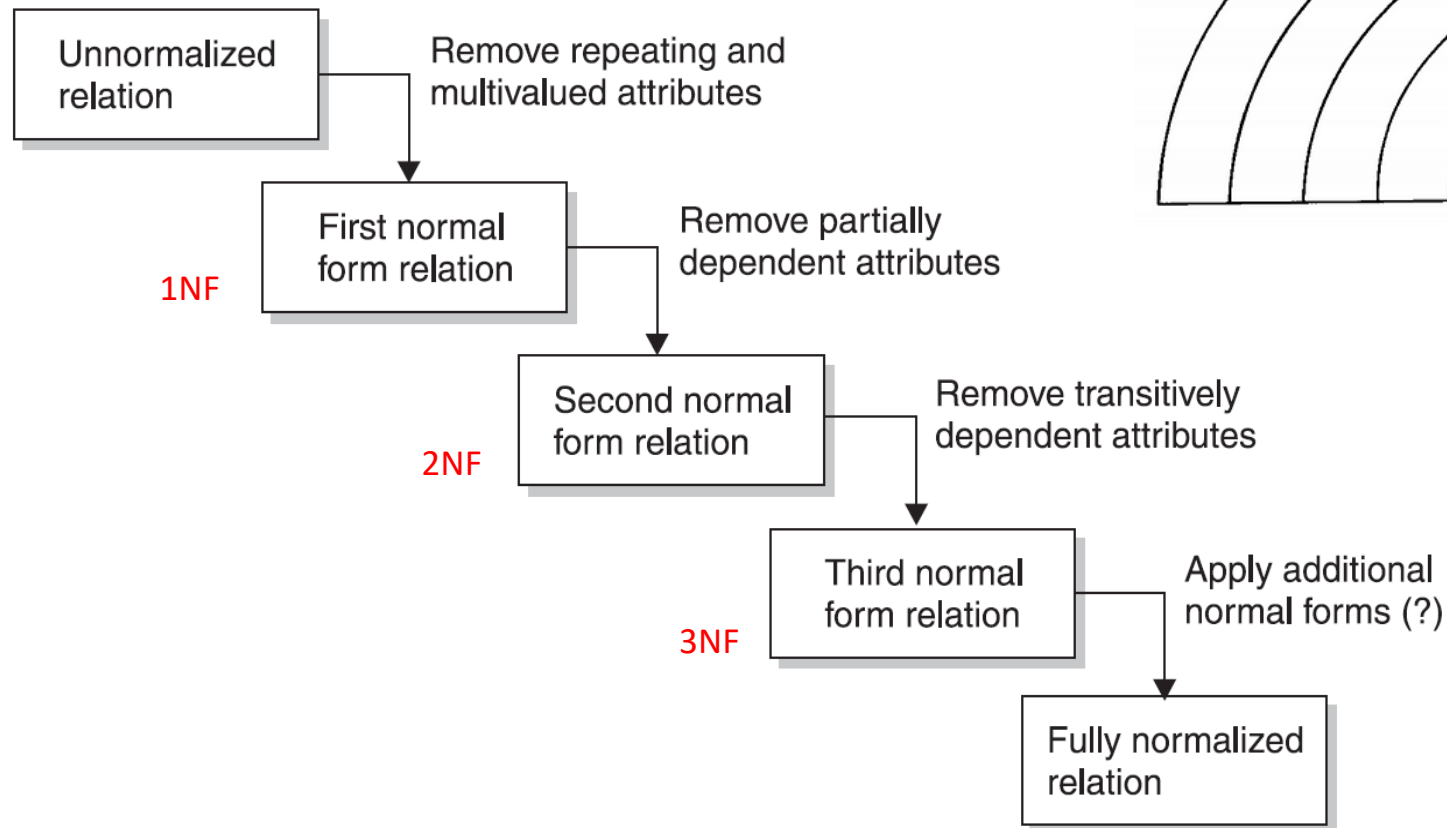
- **Atomicity** is the all or nothing requirement when making updates.
- **Consistency** requires that transactions take the database from one valid state to another.
- **Isolation** of database updates involves mechanisms that enable multiple concurrent users to simultaneously access and update the same data elements within a database.
- **Durability** ensures that any updates made by a transaction will survive a subsequent system error or problem.

ความหมายของคำศัพท์

- Key คือ ฟิลด์ (Field) แอตทริบิวต์ (Attribute) หรือ คอลัมน์ (Column) จำนวน 1 คอลัมน์ หรือ อาจเป็น หลาย ๆ คอลัมน์ มาประกอบกันก็ได้ มีค่า ไม่ซ้ำกัน ในแต่ละแถว มีคุณสมบัติที่เป็นเอกลักษณ์ (Unique) สามารถใช้คอลัมน์นั้นในการระบุถึงคอลัมน์อื่น ๆ ในแถว (Row) เรคอร์ด (Record) หรือ ทูเพิล (Tuple) นั้นได้
- Composite Key คือ คีย์ที่มีมากกว่า 1 คีย์ประกอบกัน
- Candidate Key คือ คีย์ที่มีความสามารถที่จะเป็น Primary key ได้
- Primary Key คือ คีย์หลักที่สามารถอ้างอิงแต่ละข้อมูลในตาราง ต้องมีค่าที่ไม่ซ้ำกัน และไม่มีค่าเป็น NULL (ค่าว่าง)
- Foreign Key คือ คีย์ที่เชื่อมโยงไปยัง Primary key ของตารางที่เกี่ยวข้องหรือที่มีความสัมพันธ์กัน
- Functional Dependency คือ ความสัมพันธ์ของ Primary key ที่ใช้ระบุแอตทริบิวต์อื่นโดยตรง
- Partial Dependency คือ ความสัมพันธ์ที่ใช้ Primary key แคบางส่วนในการระบุแอตทริบิวต์อื่น
- Transitive Dependency คือ ความสัมพันธ์ของแอตทริบิวต์ที่ไม่ใช่ Key แต่ใช้ระบุค่าของแอตทริบิวต์อื่นได้
- Join Dependency คือ การเชื่อมแอตทริบิวต์ที่เหมือนกันระหว่างรีเลชันที่ต่างกัน

Normalization

- กระบวนการที่ดำเนินการอย่างเป็นลำดับ เพื่อลดปัญหาการซ้ำซ้อนของข้อมูล



ทำไมต้องปรับปรุง

- เปลืองเนื้อที่เก็บข้อมูล
- ปัญหาตอนอัปเดตข้อมูล
- ปัญหาตอนเพิ่มข้อมูล
- ปัญหาตอนลบข้อมูล

Normalization

- First Normal Form: Eliminating Repeating Data

- A relation is said to be in first normal form when it contains no multivalued attributes.

ทุก ๆ field ในแต่ละ record จะเป็น single value ในตารางหนึ่ง ๆ จะไม่มีค่าของกลุ่มข้อมูลที่ซ้ำกัน

- Second Normal Form: Eliminating Partial Dependencies

- The relation is in first normal form.
- All non-key attributes are functionally dependent on the entire primary key.

ข้อมูลที่อยู่ในรีเลชันเดียวกัน ต้องขึ้นอยู่กับคีย์ใดคีย์หนึ่งเท่านั้น ไม่สามารถขึ้นอยู่กับคีย์ใดคีย์หนึ่งเพียงบางส่วนได้

- Third Normal Form: Eliminating Transitive Dependencies

- The relation is in second normal form.
- There is no transitive dependence (that is, all the non-key attributes depend only on the primary key).

คีย์แอตทริบิวต์จะต้องเป็นตัวกำหนดความหมายหรือการมีอยู่ของแอตทริบิวต์อื่น ๆ ที่อยู่ในรีเลชันเสมอ

ตัวอย่างการทำ Normalization

รหัสนักศึกษา	คำนำหน้า	ชื่อ-สกุล	รหัสแผนก	แผนก	รหัสวิชา	ชื่อวิชา
4739010001	นาย	กมลชัย เจริญดี	39	เทคโนโลยีสารสนเทศ	39011001	สารสนเทศ
					39012001	สถาปัตยกรรมคอม
4739010002	นางสาว	สุดสวຍ น้ำใจงาม	39	เทคโนโลยีสารสนเทศ	30011202	วิทยาศาสตร์ 8
					39012001	สถาปัตยกรรมคอม
					30002002	เศรษฐศาสตร์
4739010003	นาย	ประสบัติ ขยันเรียน	39	เทคโนโลยีสารสนเทศ	39011001	สารสนเทศ
					39012001	สถาปัตยกรรมคอม
4731050015	นางสาว	ไฉไล ดาวประดับ	05	อิเล็กทรอนิกส์	31052001	สายส่งอากาศ

ตัวอย่างการทำ Normalization : 1NF

รหัสนักศึกษา	คำนำหน้า	ชื่อ-สกุล	รหัสแผนก	แผนก	รหัสวิชา	ชื่อวิชา
4739010001	นาย	กมลชัย เจริญดี	39	เทคโนโลยีสารสนเทศ	39011001	สารสนเทศ
4739010001	นาย	กมลชัย เจริญดี	39	เทคโนโลยีสารสนเทศ	39012001	สถาปัตยกรรมคอม
4739010002	นางสาว	สุดสวຍ น้ำใจงาม	39	เทคโนโลยีสารสนเทศ	30011202	วิทยาศาสตร์ 8
4739010002	นางสาว	สุดสวຍ น้ำใจงาม	39	เทคโนโลยีสารสนเทศ	39012001	สถาปัตยกรรมคอม
4739010002	นางสาว	สุดสวຍ น้ำใจงาม	39	เทคโนโลยีสารสนเทศ	30002002	เศรษฐศาสตร์
4739010003	นาย	ประสพดี ขยันเรียน	39	เทคโนโลยีสารสนเทศ	39011001	สารสนเทศ
4739010003	นาย	ประสพดี ขยันเรียน	39	เทคโนโลยีสารสนเทศ	39012001	สถาปัตยกรรมคอม
4731050015	นางสาว	ไฉไล ดาวประดับ	05	อิเล็กทรอนิกส์	31052001	สายส่งอากาศ

ตัวอย่างการทำ Normalization : 2NF

รหัสนักศึกษา	คำนำหน้า	ชื่อ-สกุล	รหัสแผนก	แผนก
4739010001	นาย	กมลชัย เจริญดี	39	เทคโนโลยีสารสนเทศ
4739010002	นางสาว	สุดสวຍ น้ำใจงาม	39	เทคโนโลยีสารสนเทศ
4739010003	นาย	ประสพดี ขยันเรียน	39	เทคโนโลยีสารสนเทศ
4731050015	นางสาว	ไฉไล ดาวประดับ	05	อิเล็กทรอนิกส์

รหัสวิชา	ชื่อวิชา
39011001	สารสนเทศ
39012001	สถาปัตยกรรมคอม
30011202	วิทยาศาสตร์ 8
30002002	เศรษฐศาสตร์
31052001	สายส่งอากาศ

รหัสนักศึกษา	รหัสวิชา
4739010001	39011001
4739010001	39012001
4739010002	30011202
4739010002	39012001
4739010002	30002002
4739010003	39011001
4739010003	39012001
4731050015	31052001

ตัวอย่างการทำ Normalization : 3NF

รหัสนักศึกษา	คำนำหน้า	ชื่อ-สกุล	รหัสแผนก
4739010001	นาย	กมลชัย เจริญดี	39
4739010002	นางสาว	สุดสวຍ น้ำใจงาม	39
4739010003	นาย	ประสพดี ขยันเรียน	39
4731050015	นางสาว	ไฉไล ดาวประดับ	05

รหัสแผนก	แผนก
39	เทคโนโลยีสารสนเทศ
05	อิเล็กทรอนิกส์

รหัสวิชา	ชื่อวิชา
39011001	สารสนเทศ
39012001	สถาปัตยกรรมคอม
30011202	วิทยาศาสตร์ 8
30002002	เศรษฐศาสตร์
31052001	สายส่งอากาศ

รหัสนักศึกษา	รหัสวิชา
4739010001	39011001
4739010001	39012001
4739010002	30011202
4739010002	39012001
4739010002	30002002
4739010003	39011001
4739010003	39012001
4731050015	31052001

Example of Normalization

INVOICE: # Invoice Number, Customer Number, Customer Name, Customer Address, Customer City, Customer State, Customer Zip Code, Customer Phone, Terms, Ship Via, Order Date, (Product Number, Product Description, Quantity, Unit Price, Extended Amount), Total Order Amount



INVOICE: # Invoice Number, Customer Number, Customer Name, Customer Address, Customer City, Customer State, Customer Zip Code, Customer Phone, Terms, Ship Via, Order Date, Total Order Amount

INVOICE LINE ITEM: # Invoice Number, # Product Number, Product Description, Quantity, Unit Price, Extended Amount

1



INVOICE: # Invoice Number, Customer Number, Customer Name, Customer Address, Customer City, Customer State, Customer Zip Code, Customer Phone, Terms, Ship Via, Order Date, Total Order Amount

INVOICE LINE ITEM: # Invoice Number, # Product Number, Quantity, Sale Unit Price, Extended Amount

PRODUCT: # Product Number, Product Description, List Unit Price

2



INVOICE: # Invoice Number, Customer Number, Terms, Ship Via, Order Date

INVOICE LINE ITEM: # Invoice Number, # Product Number, Quantity, Sale Unit Price

PRODUCT: # Product Number, Product Description, List Unit Price

CUSTOMER: # Customer Number, Customer Name, Customer Address, Customer City, Customer State, Customer Zip Code, Customer Phone

3

Acme Industries INVOICE

Customer Number: 1454837

Terms: Net 30

Customer: W. Coyote

Ship Via: USPS

General Delivery

Falling Rocks, AZ 84211

Order Date: 11/05/06

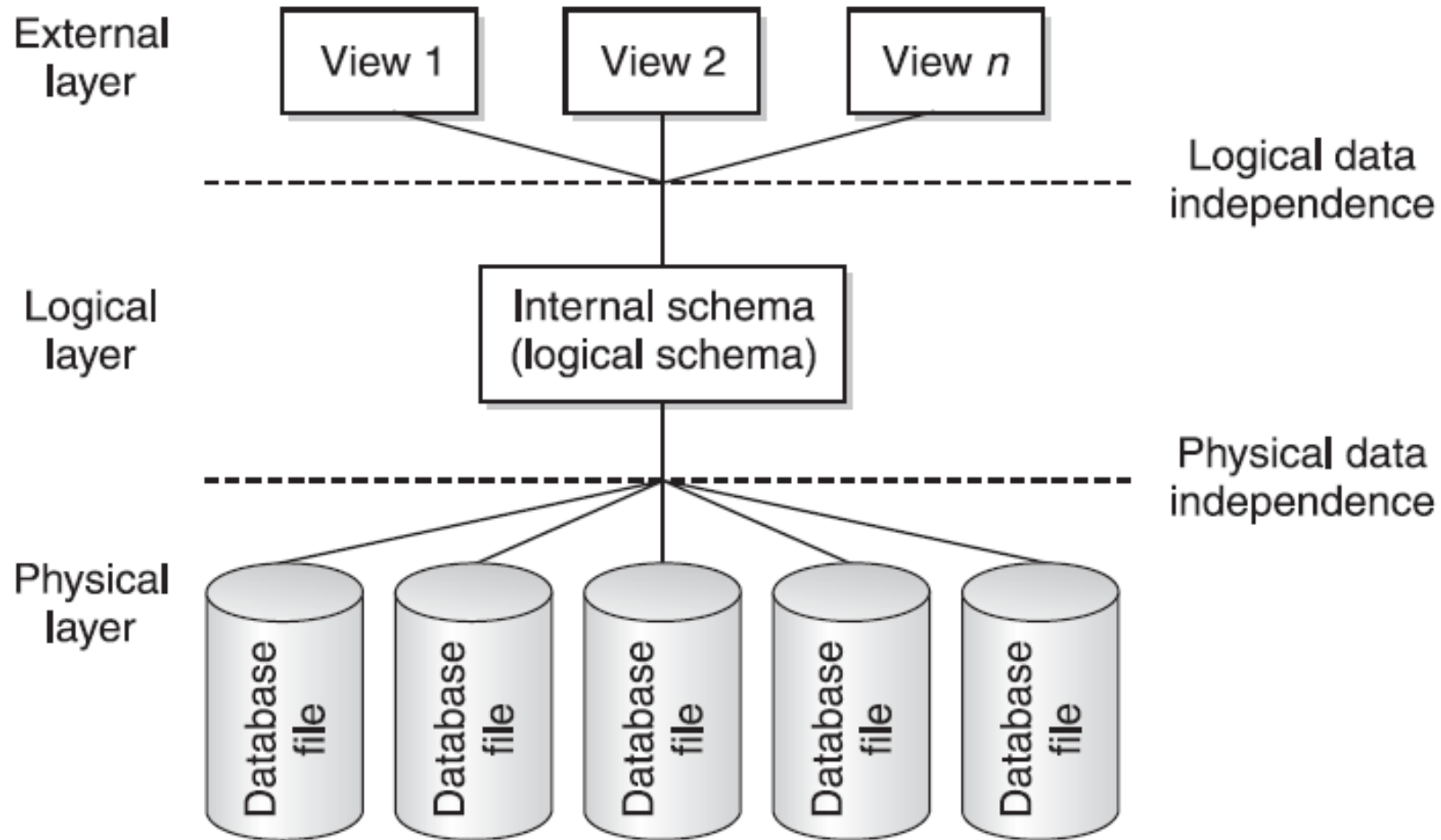
(599) 555-9345

Product No.	Description	Quantity	Unit Price	Extended Amount
SPR-2290	Super strength springs	2	24.00	\$ 48.00
STR-67	Foot straps, leather	2	2.50	\$ 5.00
HLM-45	Deluxe Crash Helmet	1	67.88	\$ 67.88
SFR-1	Rocket, solid fuel	1	128,200.40	\$ 128,200.40
ELT-7	Emergency Location Transmitter	1	79.88	** FREE GIFT **

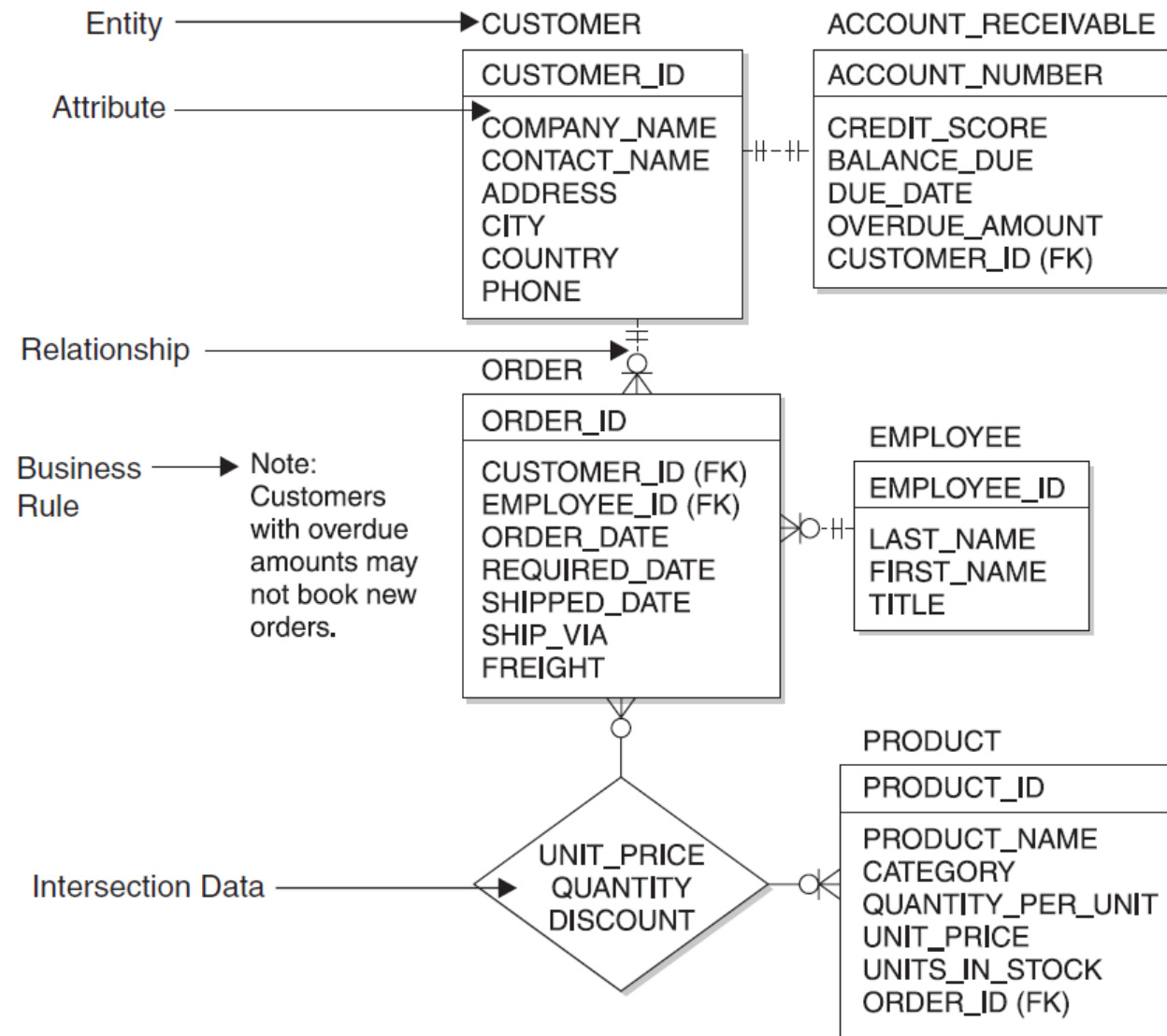
TOTAL ORDER AMOUNT:

\$ 128,321.28

Database Layers of Abstraction



Conceptual Database Design for Northwind



Example: Flat File Order System

Customer File

Customer ID	Company Name	Contact Name	Address	City	Country	Phone
ALFKI	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	Germany	030-007431
AROUT	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	UK	(171) 555-7788

Employee File

Employee ID	Last Name	First Name	Title
3	Leverling	Janet	Sales Representative
4	Peacock	Margaret	Sales Representative
6	Suyama	Michael	Sales Representative

Product File

Product ID	Product Name	Category	Quantity Per Unit	Unit Price	Units In Stock
28	Roselle Sauerkraut	Produce	25 825 g cans	\$45.60	26
39	Chartreuse verte	Beverages	750 cc per bottle	\$18.00	69
41	Jack's New England Clam Chowder	Seafood	12 12 oz cans	\$9.65	85
46	Spegesild	Seafood	4 450 g glasses	\$12.00	95
52	Filo Mix	Grains/Cereals	16 2 kg boxes	\$7.00	38
63	Veggie-spread	Condiments	15 625 g jars	\$43.90	24

Order File

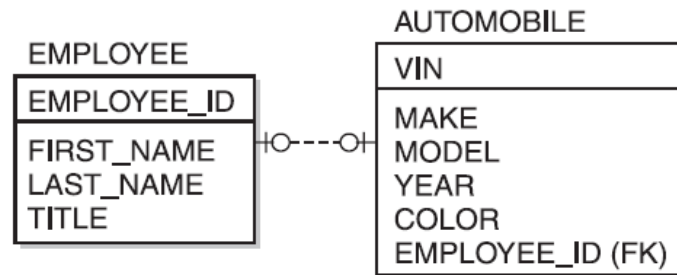
Order ID	Customer ID	Employee ID	Order Date	Required Date	Shipped Date	Ship Via	Freight
10643	ALFKI	6	25-Aug-1997	22-Sep-1997	02-Sep-1997	Speedy Express	\$29.46
10692	ALFKI	4	30-Oct-1997	03-Oct-1997	31-Oct-1997	United Package	\$61.02
10793	AROUT	3	24-Dec-1997	21-Jan-1998	08-Jan-1998	Federal Shipping	\$4.52

Order Detail File

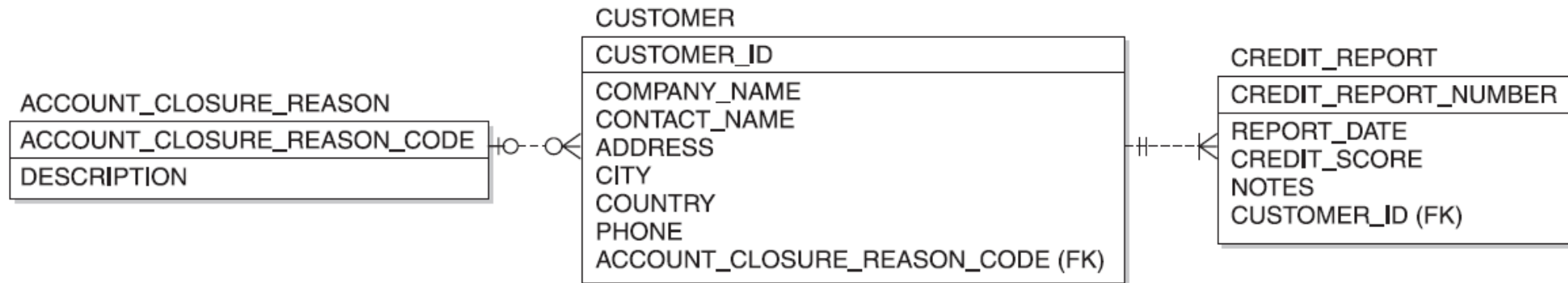
Order ID	Product ID	Unit Price	Quantity	Discount
10643	28	\$46.50	15	25%
10643	39	\$18.00	21	25%
10643	46	\$12.00	2	25%
10692	63	\$43.90	20	0%
10793	41	\$9.65	14	0%
10793	52	\$7.00	8	0%

Relationships

- One-to-One Relationships



- One-to-many relationships

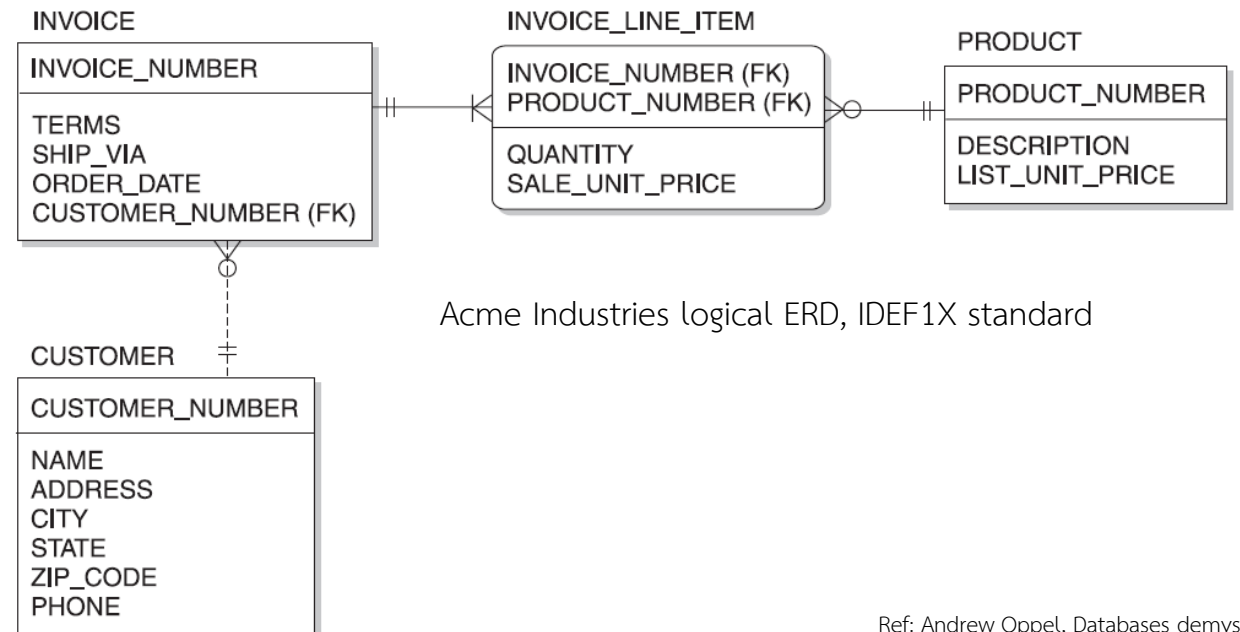
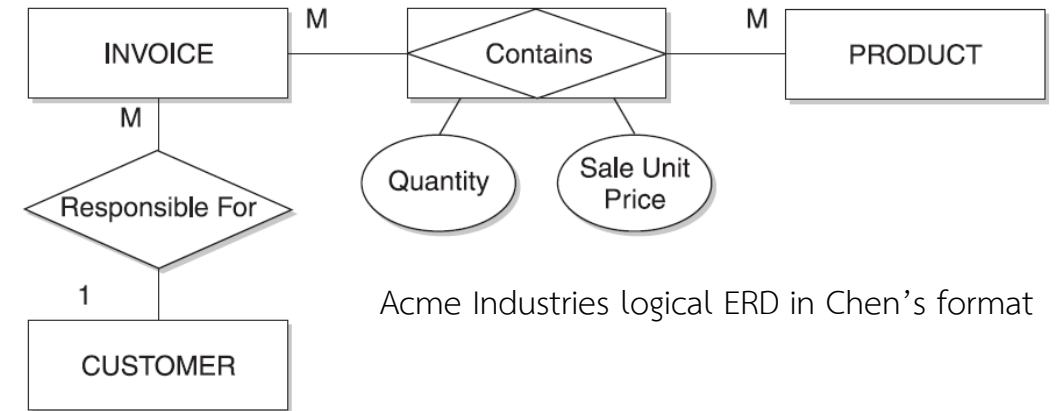
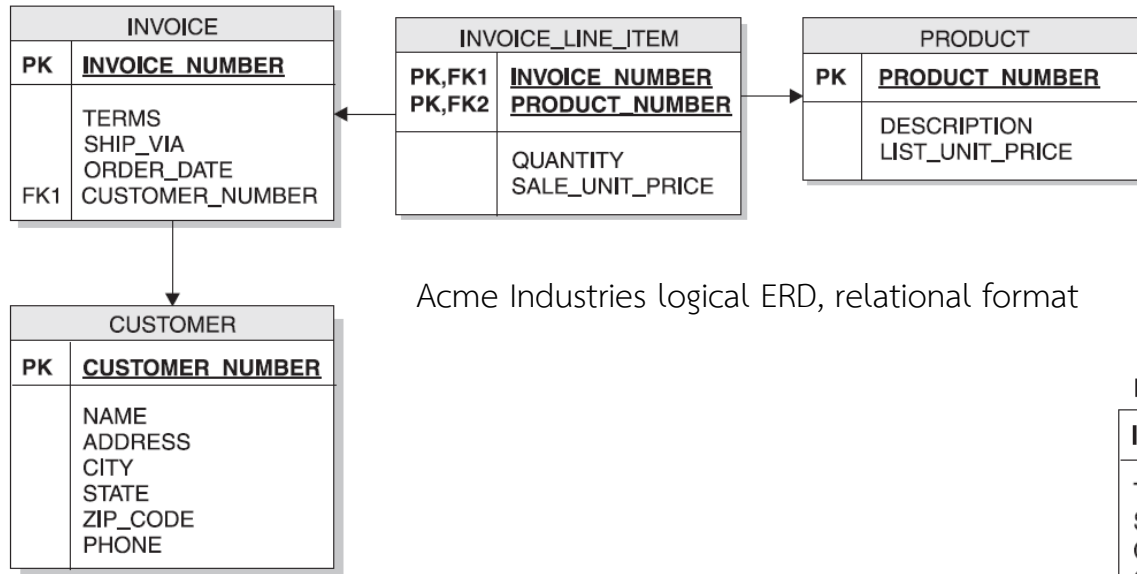


- Many-to-Many Relationships

Equivalent Data Types in Major RDBMS Products

Data Type	Microsoft Access	Microsoft SQL Server	Oracle
Fixed-Length Character	TEXT	CHAR	CHAR
Variable-Length Character	MEMO	VARCHAR	VARCHAR
Long Text	MEMO	TEXT	LONG
Integer	INTEGER or LONG INTEGER	INTEGER or SMALLINT or TINYINT	NUMBER
Decimal	NUMBER	DECIMAL or NUMERIC	NUMBER
Currency	CURRENCY	MONEY or SMALLMONEY	None, use NUMBER
Date/Time	DATE/TIME	DATETIME or SMALLDATETIME	DATE or TIMESTAMP

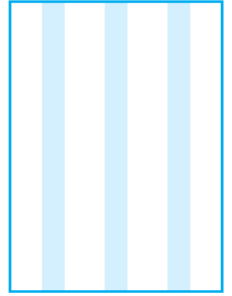
Entity Relationship Modeling



The Relational Algebra



(a) Selection



(b) Projection

P	Q	$P \times Q$														
<table><tr><th>A</th></tr><tr><td>a</td></tr><tr><td>b</td></tr></table>	A	a	b	<table><tr><th>B</th></tr><tr><td>1</td></tr><tr><td>2</td></tr><tr><td>3</td></tr></table>	B	1	2	3	=							
A																
a																
b																
B																
1																
2																
3																
		<table><tr><th>A</th><th>B</th></tr><tr><td>a</td><td>1</td></tr><tr><td>a</td><td>2</td></tr><tr><td>a</td><td>3</td></tr><tr><td>b</td><td>1</td></tr><tr><td>b</td><td>2</td></tr><tr><td>b</td><td>3</td></tr></table>	A	B	a	1	a	2	a	3	b	1	b	2	b	3
A	B															
a	1															
a	2															
a	3															
b	1															
b	2															
b	3															

(c) Cartesian product

T	
A	B
a	1
b	2

U	
B	C
1	x
1	y
3	z

$T \bowtie U$		
A	B	C
a	1	x
a	1	y

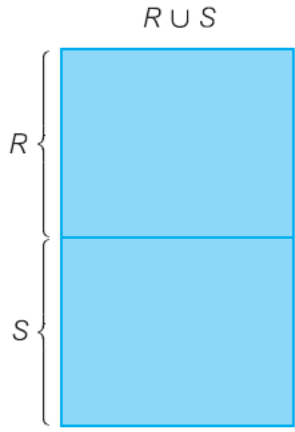
(g) Natural join

$T \triangleright_B U$	
A	B
a	1

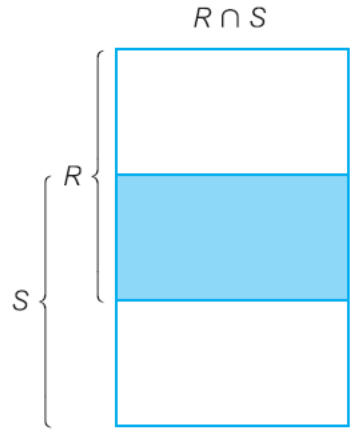
(h) Semijoin

$T \bowtie_B U$		
A	B	C
a	1	x
a	1	y
b	2	

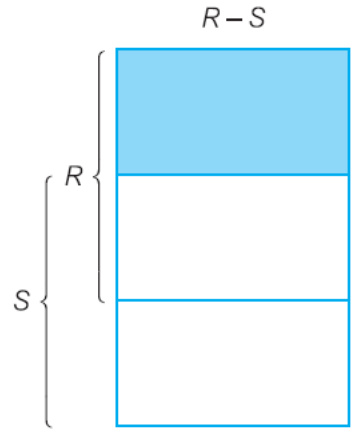
(i) Left Outer join



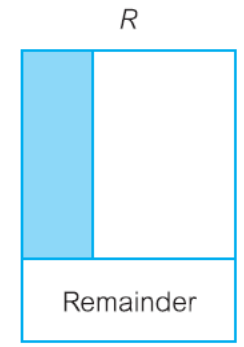
(d) Union



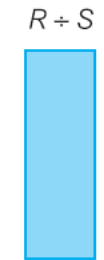
(e) Intersection



(f) Set difference



(j) Division (shaded area)



V	
A	B
a	1
a	2
b	1
b	2
c	1

W			
<table><tr><th>B</th></tr><tr><td>1</td></tr><tr><td>2</td></tr></table>	B	1	2
B			
1			
2			

$V \div W$			
<table><tr><th>A</th></tr><tr><td>a</td></tr><tr><td>b</td></tr></table>	A	a	b
A			
a			
b			

Example of division

SQL: Data Manipulation

- Ideally, a database language should allow a user to:
 - create the database and relation structures;
 - perform basic data management tasks, such as the insertion, modification, and
 - deletion of data from the relations;
 - perform both simple and complex queries.
- SQL is an example of a transform-oriented language, or a language designed to use relations to transform inputs into required outputs.
- As a language, the ISO SQL standard has two major components:
 - a Data Definition Language (DDL) for defining the database structure and controlling access to the data;
 - a Data Manipulation Language (DML) for retrieving and updating data.

SELECT – to query data in the database

INSERT – to insert data into a table

UPDATE – to update data in a table

DELETE – to delete data from a table

SQL: Data Manipulation

```
SELECT [DISTINCT | ALL] { * | [columnExpression [AS newName]] [, . . .] }  
FROM TableName [alias] [, . . .]  
[WHERE condition]  
[GROUP BY columnList] [HAVING condition]  
[ORDER BY columnList]
```

```
INSERT INTO TableName [(columnList)]  
VALUES (dataValueList)
```

```
UPDATE TableName  
SET columnName1 = dataValue1 [, columnName2 = dataValue2 . . .]  
[WHERE searchCondition]
```

```
DELETE FROM TableName  
[WHERE searchCondition]
```

SQL: Data Definition – View – Transaction – Access Control

```
CREATE TABLE TableName
    {(columnName dataType [NOT NULL] [UNIQUE]
    [DEFAULT defaultOption] [CHECK (searchCondition)] [, . . .]}
    [PRIMARY KEY (listOfColumns),]
    {[UNIQUE (listOfColumns)] [, . . .]}
    {[FOREIGN KEY (listOfForeignKeyColumns)
REFERENCES ParentTableName [(listOfCandidateKeyColumns)]
    [MATCH {PARTIAL | FULL}
    [ON UPDATE referentialAction]
    [ON DELETE referentialAction]] [, . . .]}
    {[CHECK (searchCondition)] [, . . .]}
```

```
ALTER TABLE TableName
    [ADD [COLUMN] columnName dataType [NOT NULL] [UNIQUE]
    [DEFAULT defaultOption] [CHECK (searchCondition)]]
    [DROP [COLUMN] columnName [RESTRICT | CASCADE]]
    [ADD [CONSTRAINT [ConstraintName]] tableConstraintDefinition]
    [DROP CONSTRAINT ConstraintName [RESTRICT | CASCADE]]
    [ALTER [COLUMN] SET DEFAULT defaultOption]
    [ALTER [COLUMN] DROP DEFAULT]
```

```
DROP TABLE TableName [RESTRICT | CASCADE]
```

```
CREATE [UNIQUE] INDEX IndexName
ON TableName (columnName [ASC | DESC] [, . . .])
```

```
CREATE VIEW ViewName [(newColumnName [, . . . ])]
AS subselect [WITH [CASCADE | LOCAL] CHECK OPTION]
```

```
DROP VIEW ViewName [RESTRICT | CASCADE]
```

```
SET TRANSACTION
    [READ ONLY | READ WRITE] |
    [ISOLATION LEVEL READ UNCOMMITTED | READ COMMITTED |
    REPEATABLE READ | SERIALIZABLE]
```

```
GRANT {PrivilegeList | ALL PRIVILEGES}
ON ObjectName
TO {AuthorizationIdList | PUBLIC}
[WITH GRANT OPTION]
```

Learning More

- Database Normalization ตั้งแต่ 1NF-3NF ในคลิปเดียว
<https://www.youtube.com/watch?v=FJZe3faTwGg>
- สอนพื้นฐาน SQL ทั้งหมดแบบจบในคลิปเดียว !!
<https://www.youtube.com/watch?v=vd1qdnCX5RU>
- สอนพื้นฐาน JavaScript ทั้งหมดแบบจบในคลิปเดียว !!
<https://www.youtube.com/watch?v=PGZ7QiKdumo>

SQL

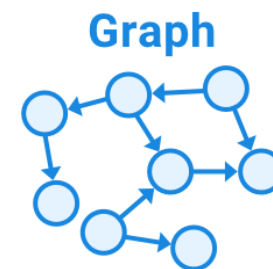
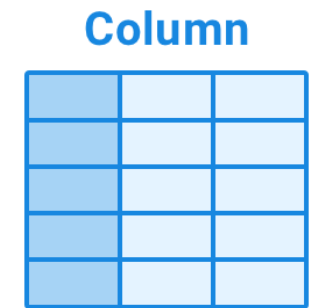
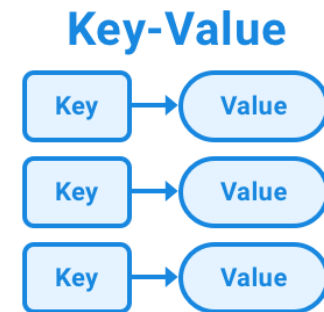
- Structured Query Language (SQL) is a programming language that's been around since the early 1970s. Back then, data storage was expensive, so a key focus of SQL was to cut down on the duplication of data.
- SQL is widely used for data management in relational database management systems.
- The language is used to query relational databases, which are databases that recognize relationships between stored data items.
- Examples of SQL :

MySQL	CockroachDB
Oracle	Azure SQL Database
Microsoft SQL Server	Microsoft Access
PostgreSQL	Ingres
Db2	Sybase
YugabyteDB	SQLite

NoSQL

- NoSQL databases are non-relational (as opposed to SQL, which is relational). They first emerged in the late 2000s when developer productivity became more important than storage costs.
- NoSQL does not mean that these databases never use SQL. (There are NoSQL databases that can and do support SQL.) Instead, it's best to think of NoSQL as “not only SQL.”
- Examples of NoSQL :

Google Cloud BigTable	Neo4j
Apache HBase	HBase
Redis	InfinityDB
FaunaDB	DynamoDB
CouchDB	MariaDB
MongoDB	Scylla
Cassandra	ArangoDB
Elasticsearch	InfiniteGraph



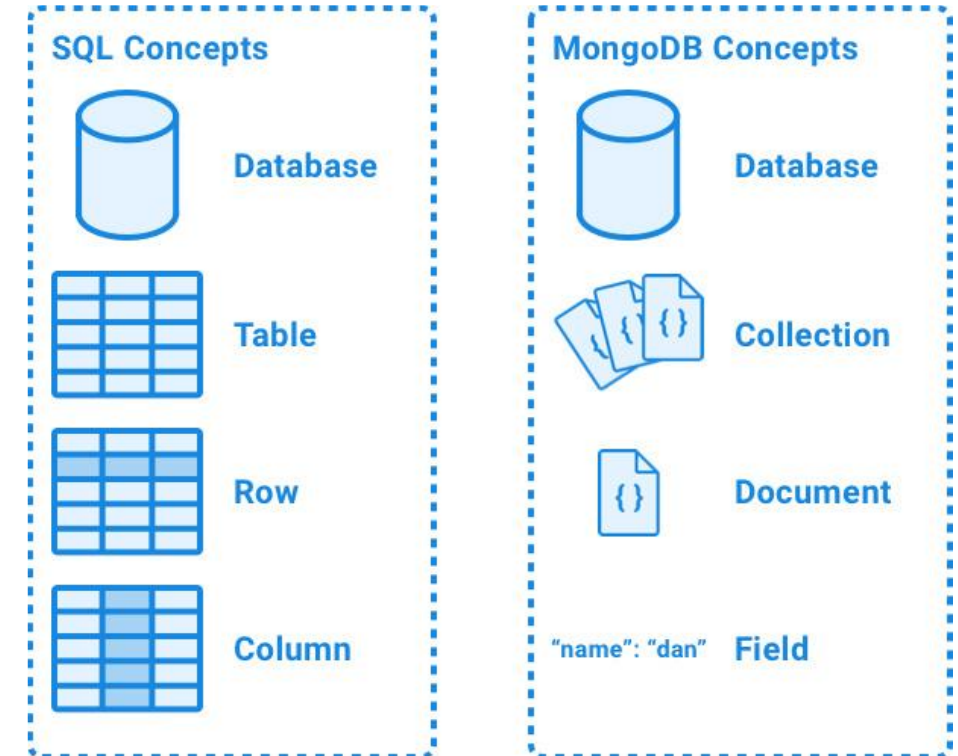
SQL vs. NoSQL

	SQL Databases	NoSQL Databases
Language	SQL databases use structured query languages to perform operations, requiring the use of predefined schema to better interact with the data.	On the other hand, NoSQL databases use a dynamic schema to query data. Also, some NoSQL databases use SQL-like syntax for document manipulation.
Data Schema	SQL databases have a predefined and fixed format, which cannot be changed for new data.	NoSQL databases are more flexible. This flexibility means that records in the databases can be created without having a predefined structure, and each record has its own structure.
Scalability	SQL databases are only vertically scalable, meaning that a single machine needs to increase CPU, RAM, SSD, at a certain level to meet the demand.	NoSQL databases are horizontally scalable, meaning that additional machines are added to the existing infrastructure to satisfy the storage demand.
Big Data Support	The vertical scaling makes it difficult for SQL databases to store very big data (petabytes).	The horizontal scaling and dynamic data schema make NoSQL suitable for big data. Also, NoSQL databases were developed by top internet companies (Amazon, Google, Yahoo, etc.) to face the challenges of the rapidly increasing amount of data.
Properties	SQL databases use the ACID (Atomicity, Consistency, Isolation, Durability) property.	NoSQL databases, on the other hand, use the CAP (Consistency, Availability, Partition Tolerance) property.

MongoDB

- MongoDB is an open-source document-oriented database that stores data in JSON format, called BSON.
- It is the most commonly used database and was designed for high availability and scalability, providing auto-sharing and built-in replication.
- A record in MongoDB is a document, which is a data structure composed of key value pairs similar to the structure of JSON objects.

```
{  
  title: "Post Title 1",  
  body: "Body of post.",  
  category: "News",  
  likes: 1,  
  tags: ["news", "events"],  
  date: Date()  
}
```



Daniel Bugl, Modern Full-Stack React Projects

Fun with MySQL + Node.js

- Install MySQL Server from <https://dev.mysql.com/downloads/mysql/>
- Add PATH to environment
- Connect to mysql :

```
mysql -u root -p
```

- For compatibility issues in MySQL 8.0 or later, do the following :

```
ALTER USER 'root' @'localhost' IDENTIFIED WITH mysql_native_password BY 'password';
```

- Install MySQL Driver for Node.js :

```
npm install mysql2
```

- Node.js can use this module to manipulate the MySQL database :

```
var mysql = require("mysql2");

var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "password",
});

con.connect(function (err) {
  if (err) throw err;
  console.log("Connected Success!");
});
```

- Create Database

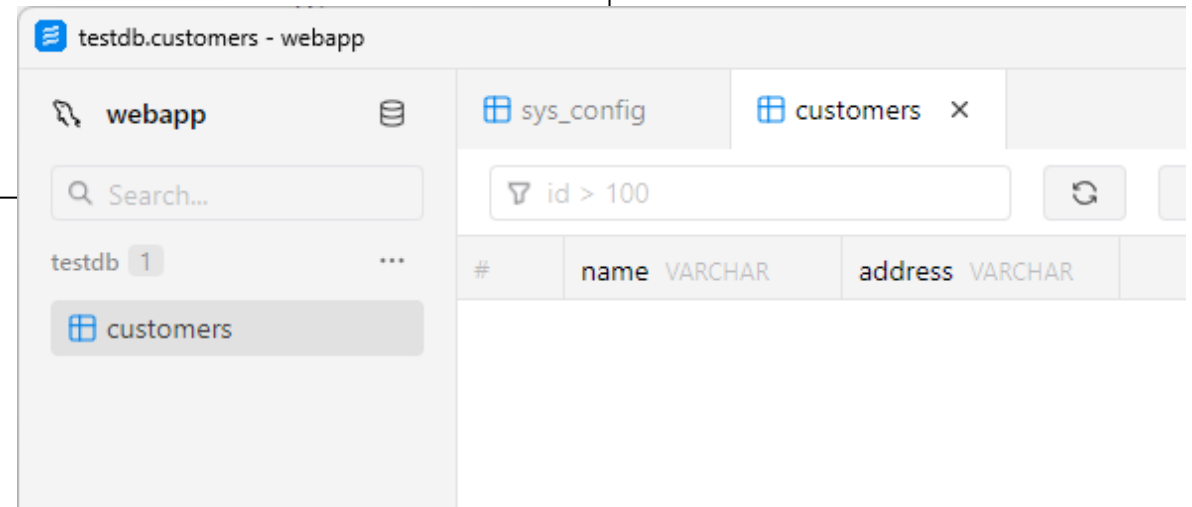
```
con.connect(function (err) {
  if (err) throw err;
  console.log("Connected Success!");
  con.query("CREATE DATABASE testdb", function (err, result) {
    if (err) throw err;
    console.log("Database created");
  });
});
```

- Create Table

```
var mysql = require("mysql2");

var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "password",
  database: "testdb",
});

con.connect(function (err) {
  if (err) throw err;
  console.log("Connected!");
  var sql = "CREATE TABLE customers (name VARCHAR(255), address VARCHAR(255))";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("Table created");
  });
});
```



- Primary Key

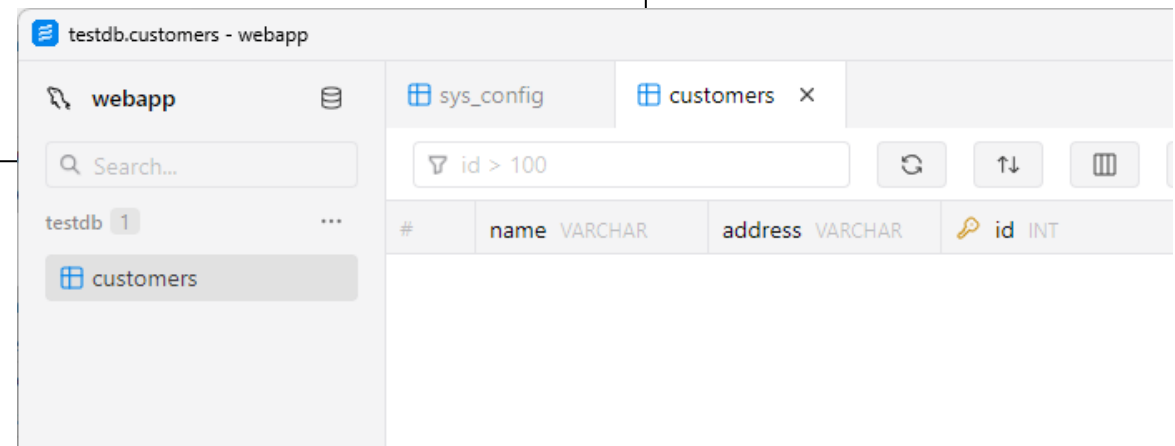
```
var sql = "CREATE TABLE customers (id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(255), address VARCHAR(255))";
```

- Alter Table

```
var mysql = require("mysql2");

var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "password",
  database: "testdb",
});

con.connect(function (err) {
  if (err) throw err;
  console.log("Connected!");
  var sql = "ALTER TABLE customers ADD COLUMN id INT AUTO_INCREMENT PRIMARY KEY";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("Table altered");
  });
});
```



- Insert data

```
var mysql = require("mysql2");

var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "password",
  database: "testdb",
});

con.connect(function (err) {
  if (err) throw err;
  console.log("Connected!");
  var sql = "INSERT INTO customers (name, address) VALUES ('Company Inc', 'Highway 37')";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("1 record inserted");
  });
});
```

testdb.customers - webapp

webapp

Search...

testdb 1

customers

sys_config customers x

id > 100

#	name VARCHAR	address VARCHAR	id INT
1	Company Inc	Highway 37	1

- Insert multiple data

```
var mysql = require("mysql2");

var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "password",
  database: "testdb",
});

con.connect(function (err) {
  if (err) throw err;
  console.log("Connected!");
  var sql = "INSERT INTO customers (name, address) VALUES ?";
  var values = [
    ["John", "Highway 71"],
    ["Peter", "Lowstreet 4"],
    ["Amy", "Apple st 652"],
    ["Hannah", "Mountain 21"],
    ["Michael", "Valley 345"],
    ["Sandy", "Ocean blvd 2"],
    ["Betty", "Green Grass 1"],
    ["Richard", "Sky st 331"],
    ["Susan", "One way 98"],
    ["Vicky", "Yellow Garden 2"],
    ["Ben", "Park Lane 38"],
    ["William", "Central st 954"],
    ["Chuck", "Main Road 989"],
    ["Viola", "Sideway 1633"],
  ];

  con.query(sql, [values], function (err, result) {
    if (err) throw err;
    console.log("Number of records inserted: " + result.affectedRows);
  });
});
```

testdb.customers - webapp

webapp

Search...

testdb 1

customers

sys_config customers x

id > 100

#	name VARCHAR	address VARCHAR	id INT
1	Company Inc	Highway 37	1
2	John	Highway 71	2
3	Peter	Lowstreet 4	3
4	Amy	Apple st 652	4
5	Hannah	Mountain 21	5
6	Michael	Valley 345	6

- Selecting from a table

```
var mysql = require("mysql2");

var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "password",
  database: "testdb",
});

con.connect(function (err) {
  if (err) throw err;
  con.query("SELECT * FROM customers", function (err, result, fields) {
    if (err) throw err;
    console.log(result);
  });
});
```

"SELECT name, address FROM customers"

```
C:\Users\Suriya\Documents\Code\webapp\04-database>node demo_db_select.js
[
  RowDataPacket { name: 'Company Inc', address: 'Highway 37' },
  RowDataPacket { name: 'John', address: 'Highway 71' },
  RowDataPacket { name: 'Peter', address: 'Lowstreet 4' },
  RowDataPacket { name: 'Amy', address: 'Apple st 652' },
  RowDataPacket { name: 'Hannah', address: 'Mountain 21' },
  RowDataPacket { name: 'Michael', address: 'Valley 345' },
  RowDataPacket { name: 'Sandy', address: 'Ocean blvd 2' },
  RowDataPacket { name: 'Betty', address: 'Green Grass 1' },
  RowDataPacket { name: 'Richard', address: 'Sky st 331' },
  RowDataPacket { name: 'Susan', address: 'One way 98' },
  RowDataPacket { name: 'Vicky', address: 'Yellow Garden 2' },
  RowDataPacket { name: 'Ben', address: 'Park Lane 38' },
  RowDataPacket { name: 'William', address: 'Central st 954' },
  RowDataPacket { name: 'Chuck', address: 'Main Road 989' },
  RowDataPacket { name: 'Viola', address: 'Sideway 1633' }
]
```

```
C:\Users\Suriya\Documents\Code\webapp\04-database>node demo_db_select.js
[
  RowDataPacket { name: 'Company Inc', address: 'Highway 37', id: 1 },
  RowDataPacket { name: 'John', address: 'Highway 71', id: 2 },
  RowDataPacket { name: 'Peter', address: 'Lowstreet 4', id: 3 },
  RowDataPacket { name: 'Amy', address: 'Apple st 652', id: 4 },
  RowDataPacket { name: 'Hannah', address: 'Mountain 21', id: 5 },
  RowDataPacket { name: 'Michael', address: 'Valley 345', id: 6 },
  RowDataPacket { name: 'Sandy', address: 'Ocean blvd 2', id: 7 },
  RowDataPacket { name: 'Betty', address: 'Green Grass 1', id: 8 },
  RowDataPacket { name: 'Richard', address: 'Sky st 331', id: 9 },
  RowDataPacket { name: 'Susan', address: 'One way 98', id: 10 },
  RowDataPacket { name: 'Vicky', address: 'Yellow Garden 2', id: 11 },
  RowDataPacket { name: 'Ben', address: 'Park Lane 38', id: 12 },
  RowDataPacket { name: 'William', address: 'Central st 954', id: 13 },
  RowDataPacket { name: 'Chuck', address: 'Main Road 989', id: 14 },
  RowDataPacket { name: 'Viola', address: 'Sideway 1633', id: 15 }
]
```

"SELECT * FROM customers LIMIT 5"

- Select with a filter

```
"SELECT * FROM customers WHERE address = 'Park Lane 38'"
```

```
"SELECT * FROM customers WHERE address LIKE 'S%'"
```

- Sort the result

```
"SELECT * FROM customers ORDER BY name"
```

- Delete record

```
"DELETE FROM customers WHERE address = 'Mountain 21'"
```

- Drop the table

```
"DROP TABLE customers"
```

- Update table

```
"UPDATE customers SET address = 'Canyon 123' WHERE address = 'Valley 345'"
```

- Join table

```
"SELECT users.name AS user, products.name AS favorite FROM users JOIN products ON users.favorite_product = products.id"
```

Fun with MongoDB + Node.js

- Install MongoDB Community Server from <https://www.mongodb.com/try/download/community>
- Download and extract Mongosh from <https://www.mongodb.com/try/download/shell>
- Add PATH to environment
- For MongoDB's security, add the user and password :

```
use admin
db.createUser({user:"foouser",pwd:"foopwd",roles:[{role:"root",db:"admin"}]})
```

Insert Data with MongoDB Compass

The following steps illustrate how to insert data into a MongoDB collection using MongoDB Compass:

- Main Interface:** The MongoDB Compass interface shows the connection to `localhost:27017`. The **Databases** tab is selected, showing a list of databases. A red arrow points from the **+ Create database** button to the next step.
- Create Database:** The **Create Database** dialog is shown. The **Database Name** is `FIBOSStudentsDB` and the **Collection Name** is `FIBOSStudentCollection`. The **Time-Series** checkbox is unchecked. A red arrow points from the **Create Database** button to the next step.
- FIBOSStudentCollection Overview:** The **FIBOSStudentCollection** overview is shown. It displays the **Storage size** (4.10 kB) and the number of **Documents** (1). A red arrow points from the **Documents** section to the next step.
- Insert Document:** The **Insert Document** dialog is shown. It displays the document structure for the `FIBOSStudentCollection`. The document contains the following fields:

```
{
  "_id": ObjectId('67aeff51c1f43fe18dc2f2fc'),
  "studentId": "",
  "studentName": ""
}
```

A red arrow points to the **studentName** field, indicating where to enter the data. The **Insert** button is visible at the bottom right.

- Install MongoDB Driver for Node.js :

```
npm install mongodb
```

- Connect and create database :

```
const { MongoClient } = require("mongodb");
// or as an es module:
// import { MongoClient } from 'mongodb'

// Connection URL
const url = "mongodb://localhost:27017";
const client = new MongoClient(url);

// Database Name
const dbName = "myProject";

async function main() {
  // Use connect method to connect to the server
  await client.connect();
  console.log("Connected successfully to server");
  const db = client.db(dbName);
  const collection = db.collection("documents");

  // the following code examples can be pasted here...

  return "done.";
}

main()
  .then(console.log)
  .catch(console.error)
  .finally(() => client.close());
```

- Insert a new document

```
const insertResult = await collection.insertMany([ { a: 1 }, { a: 2 }, { a: 3 } ]);
console.log("Inserted documents =>", insertResult);
```

- Find all documents

```
const findResult = await collection.find({}).toArray();
console.log("Found documents =>", findResult);
```

- Find documents with a query filter

```
collection.findOne({ a: 3 })
```

```
const filteredDocs = await collection.find({ a: 3 }).toArray();
console.log("Found documents filtered by { a: 3 } =>\n", filteredDocs);
```

- Update a document

```
const updateResult = await collection.updateOne({ a: 3 }, { $set: { b: 1 } });
console.log("Updated documents =>", updateResult);
```

- Remove a document

```
collection.deleteOne({ a: 3 })
```

```
const deleteResult = await collection.deleteMany({ a: 3 });
console.log("Deleted documents =>", deleteResult);
```

- Index a collection

```
const indexName = await collection.createIndex({ a: 1 });
console.log("index name =", indexName);
```

Dataflare

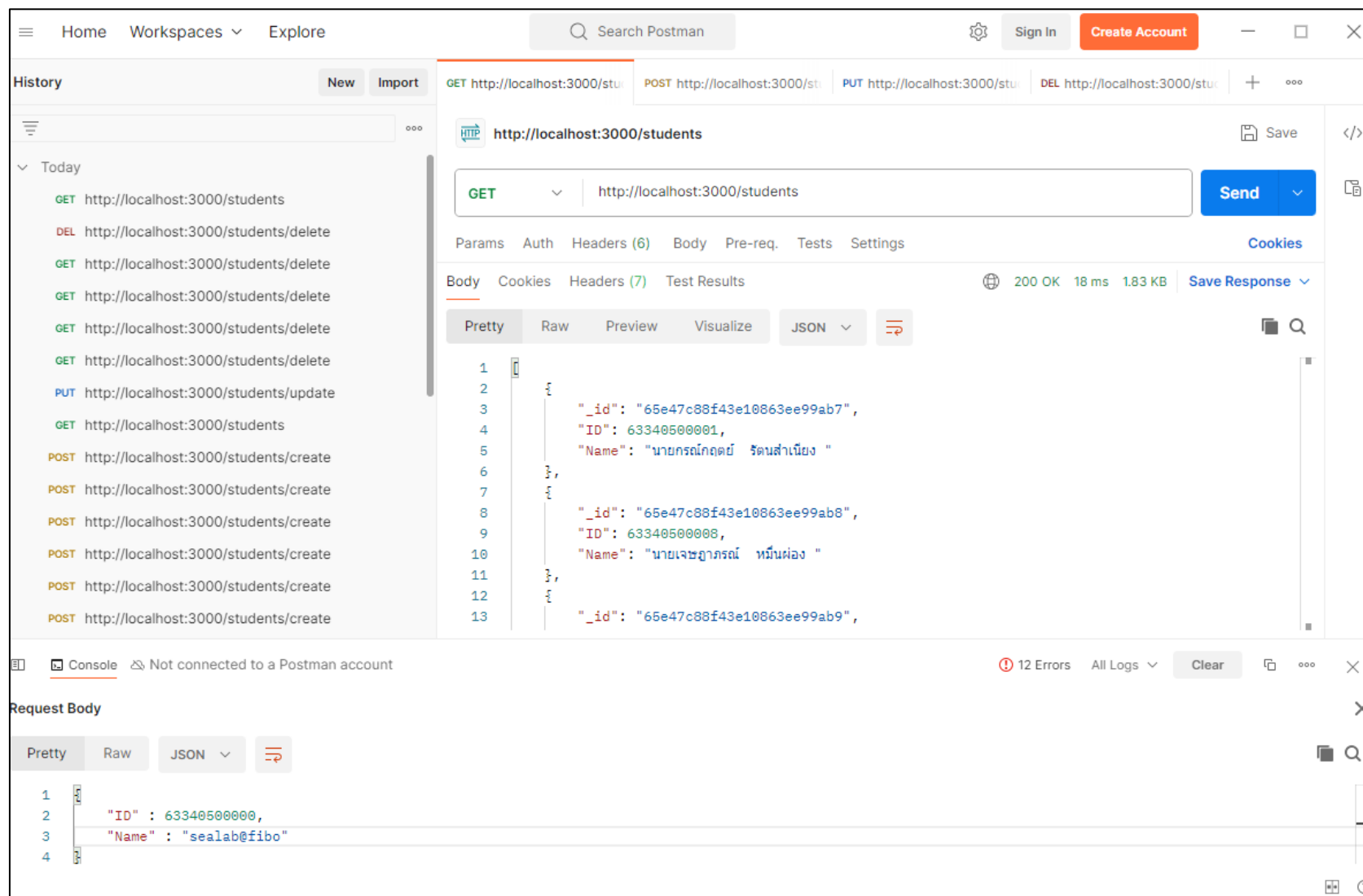
Easily manage your Table,
view Data, write SQL and
run Query

The screenshot shows the Dataflare web application interface. At the top, the Dataflare logo is on the left and a Twitter icon is on the right. The main heading is "Manage Your Database Better" in a large, white, sans-serif font. Below this, a subtitle reads "Easily manage your Table, view Data, write SQL and run Query." in a smaller, lighter font. A prominent blue button with white text says "Try Dataflare for free". Below the button are three small icons: a database cylinder, a table grid, and a flame. A navigation bar contains four buttons: "View Table" (highlighted), "SQL Query", "Schema Manager", and "Create Table", followed by a settings gear icon. The bottom section is a dark-themed sidebar and main content area. The sidebar on the left is titled "Docker MySQL" and has a search bar. Below it, a list of database components is shown, with "mysql 38" selected. The main content area displays a table titled "help_topic" with a filter "help_topic_id > 3". The table has columns: #, help..., name CHAR, help..., descriptio..., example T..., and url TEXT. It contains five rows of data.

#	help...	name CHAR	help...	descriptio...	example T...	url TEXT
1	4	ASYMMETRIC_DEC...	5	Syntax: asymmetric..	EMPTY	https://dev.r
2	5	ASYMMETRIC_ENC...	5	Syntax: asymmetric..	-- Generate private/...	https://dev.r
3	6	ASYMMETRIC_SIGN	5	Syntax: asymmetric..	EMPTY	https://dev.r
4	7	ASYMMETRIC_VERI...	5	Syntax: asymmetric..	-- Set the encryptio...	https://dev.r
5	8	CREATE_ASYMMET...	5	Syntax: create_asy...	SET @priv = create...	https://dev.r

Postman

Postman is an API platform for building and using APIs. Postman simplifies each step of the API lifecycle and streamlines collaboration so you can create better APIs—faster.



MongoDB with React

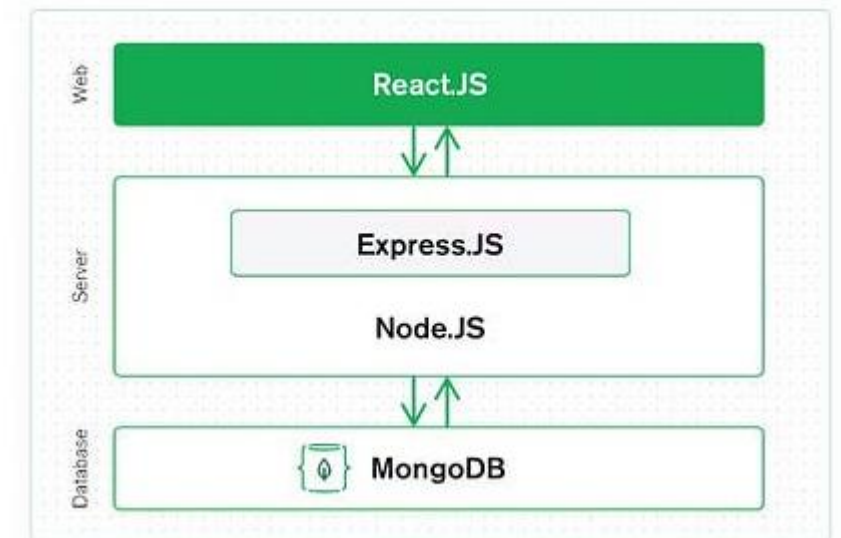
```
const express = require("express");
const app = express();
const port = 3000;
const { MongoClient } = require("mongodb");
const url = "mongodb://localhost:27017";

app.use(express.json());

app.get("/", function (req, res) {
  res.send("Hello World!");
});

app.get("/students", async (req, res) => {
  const id = parseInt(req.params.id);
  const client = new MongoClient(url);
  await client.connect();
  const students = await client
    .db("myDB")
    .collection("student")
    .find({})
    .toArray();
  await client.close();
  res.status(200).send(students);
});

app.listen(port, function () {
  console.log(`Example app listening on port ${port}!`);
});
```

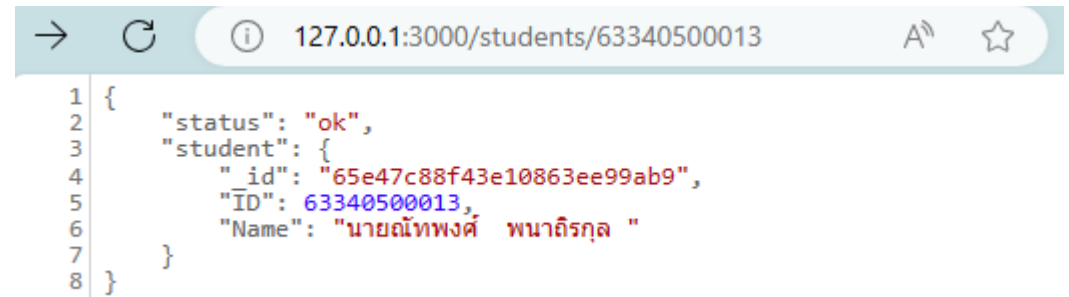


Ref: <https://medium.com/@kaklotarrahul79/step-by-step-guide-connecting-mongodb-with-react-js-for-seamless-full-stack-development-db51c34da282>

Ref: <https://karnyong.medium.com/%E0%B9%80%E0%B8%82%E0%B8%B5%E0%B8%A2%E0%B8%99-api-%E0%B8%AA%E0%B8%B3%E0%B8%AB%E0%B8%A3%E0%B8%B1%E0%B8%9A-crud-%E0%B8%82%E0%B9%89%E0%B8%AD%E0%B8%A1%E0%B8%B9%E0%B8%A5-%E0%B8%94%E0%B9%89%E0%B8%A7%E0%B8%A2-express-js-%E0%B9%81%E0%B8%A5%E0%B8%B0-mongodb-62b6b799b280>

MongoDB with React: Read by id API

```
app.get("/students/:id", async (req, res) => {
  const id = parseInt(req.params.id);
  const client = new MongoClient(url);
  await client.connect();
  const student = await client
    .db("myDB")
    .collection("student")
    .findOne({ ID: id });
  await client.close();
  res.status(200).send({ status: "ok", student: student });
});
```



```
1 {
2   "status": "ok",
3   "student": {
4     "id": "65e47c88f43e10863ee99ab9",
5     "ID": 63340500013,
6     "Name": "นายณัฏพวงศ์ พนาธิรกุล "
7   }
8 }
```

MongoDB with React: Create API

```
app.post("/students/create", async (req, res) => {
  const student = req.body;
  const client = new MongoClient(url);
  await client.connect();
  await client
    .db("myDB")
    .collection("student")
    .insertOne({
      ID: parseInt(student.ID),
      Name: student.Name,
    });
  await client.close();
  res.status(200).send({
    status: "ok",
    message: "Student with ID = " + student.ID + " is created",
    student: student,
  });
});
```

MongoDB with React: Update API

```
app.put("/students/update", async (req, res) => {
  const student = req.body;
  const id = parseInt(student.ID);
  const client = new MongoClient(url);
  await client.connect();
  await client
    .db("myDB")
    .collection("student")
    .updateOne(
      { ID: id },
      {
        $set: {
          ID: parseInt(student.ID),
          Name: student.Name,
        },
      }
    );
  await client.close();
  res.status(200).send({
    status: "ok",
    message: "Student with ID = " + id + " is updated",
    Student: student,
  });
});
```

MongoDB with React: Delete API

```
app.delete("/students/delete", async (req, res) => {
  const id = parseInt(req.body.ID);
  const client = new MongoClient(url);
  await client.connect();
  await client.db("myDB").collection("student").deleteOne({ ID: id });
  await client.close();
  res.status(200).send({
    status: "ok",
    message: "Student with ID = " + id + " is deleted",
  });
});
```

Mongoose

- Mongoose is an elegant Object Data Modeling (ODM) library built for MongoDB and JavaScript.
- Mongoose bridges the gap between the application and the MongoDB database.
- It offers a schema-based solution to model and structure data, providing a structured and organized way to interact with MongoDB.
- Key features: Schema definition, Validation, Middleware and Hooks, Query Building, Population, Data Casting

- Install Mongoose: `npm install mongoose`

```
const schema = new Schema({
  name: String,
  binary: Buffer,
  living: Boolean,
  updated: { type: Date, default: Date.now() },
  age: { type: Number, min: 18, max: 65, required: true },
  mixed: Schema.Types.Mixed,
  _someId: Schema.Types.ObjectId,
  array: [],
  ofString: [String],
  nested: { stuff: { type: String, lowercase: true, trim: true } },
});
```

Connecting to MongoDB

```
// Import the mongoose module
const mongoose = require("mongoose");

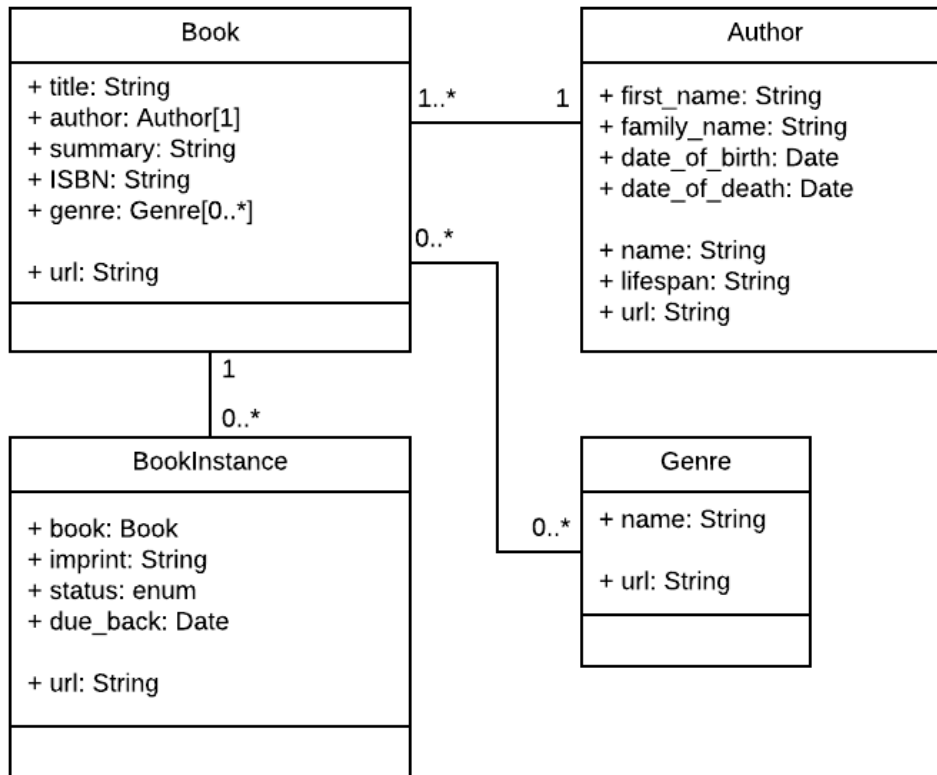
// Set `strictQuery: false` to globally opt into filtering by properties that aren't in the schema
// Included because it removes preparatory warnings for Mongoose 7.
// See: https://mongoosejs.com/docs/migrating_to_6.html#strictquery-is-removed-and-replaced-by-strict
mongoose.set("strictQuery", false);

// Define the database URL to connect to.
const mongoDB = "mongodb://127.0.0.1/my_database";

// Wait for database to connect, logging an error if there is a problem
main().catch((err) => console.log(err));
async function main() {
  await mongoose.connect(mongoDB);
}
```

Define Mongoose Schema and Model

- A schema-based Mongoose model acts as a direct interface to a MongoDB collection in a database.
- All interactions with the database will happen through the model.



./models/book.js

```
const mongoose = require("mongoose");

const Schema = mongoose.Schema;

const BookSchema = new Schema({
  title: { type: String, required: true },
  author: { type: Schema.Types.ObjectId, ref: "Author", required: true },
  summary: { type: String, required: true },
  isbn: { type: String, required: true },
  genre: [{ type: Schema.Types.ObjectId, ref: "Genre" }],
});

// Virtual for book's URL
BookSchema.virtual("url").get(function () {
  // We don't use an arrow function as we'll need the this object
  return `/catalog/book/${this._id}`;
});

// Export model
module.exports = mongoose.model("Book", BookSchema);
```


CRUD Operations

CRUD (Create, Read, Update และ Delete) บนฐานข้อมูล

- Create : Add ข้อมูลใหม่ในฐานข้อมูล
- Read : แสดงผลข้อมูลที่มีอยู่ในฐานข้อมูล
- Update : แก้ไขข้อมูลที่มีอยู่ในฐานข้อมูล
- Delete : ลบข้อมูลที่มีอยู่ในฐานข้อมูล

Fun 05: MySQL and MongoDB

- ให้ปรับเว็บไซต์ Fun 04 โดยให้เพิ่มฐานข้อมูลเพื่อเก็บข้อมูลการนับ โดยเลือกใช้ MySQL หรือ MongoDB และออกแบบ Database ให้เหมาะสม
- เมื่อกด Save จะทำการบันทึกค่าจำนวนที่นับได้แยกชาย หญิง ลงในฐานข้อมูล พร้อมวันที่บันทึก และรีเซ็ตจำนวนเป็นศูนย์
- ดึงข้อมูลจากฐานข้อมูลมาแสดงผลการนับที่สามารถเลือกช่วงเวลาได้