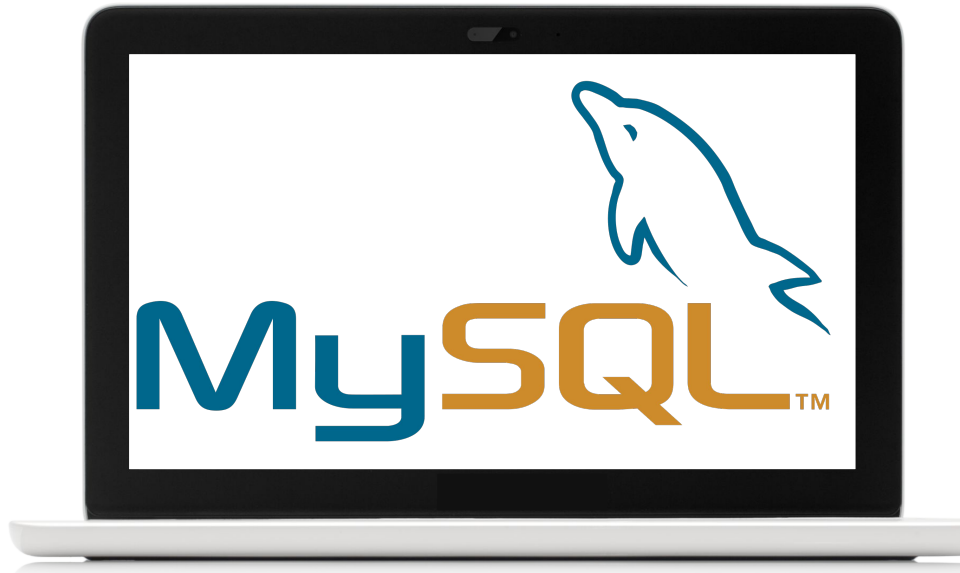


Databases and MySQL

Databases are a structured set of data. MySQL is an open-source relational database management system.



Outline



1. Intro to Databases
2. ER Model
3. Relationships
4. Normalization
5. MySQL
 - Data Types & Constraints
 - DDL vs DML
 - DQL/DRL
 - Subqueries
 - Joins
 - Views
 - Procedures & Cursors
 - Stored Functions

Intro to Databases



What is a Database?

A **database** is a collection of information organized in such a way that a computer program can quickly select desired pieces of data.



Databases

- Data stored on backend database server
- Database types:
 - ◆ *Flat File*
 - ◆ *Hierarchical*
 - ◆ *Network*
 - ◆ *Relational*
 - ◆ *Object Oriented*
 - ◆ *Object Relational*

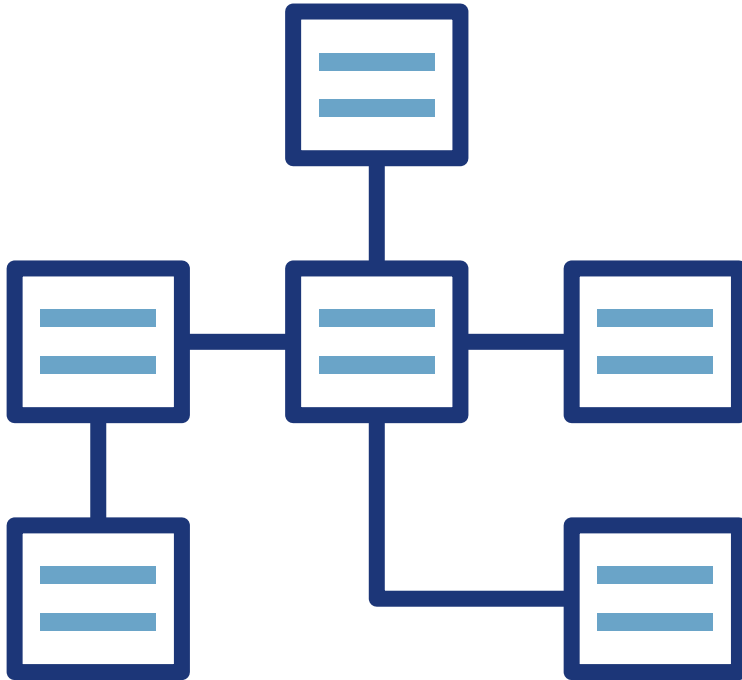


Database Management System (DBMS)

A **DBMS** helps you create and manage databases—like MS Word helps you create and manage word documents.



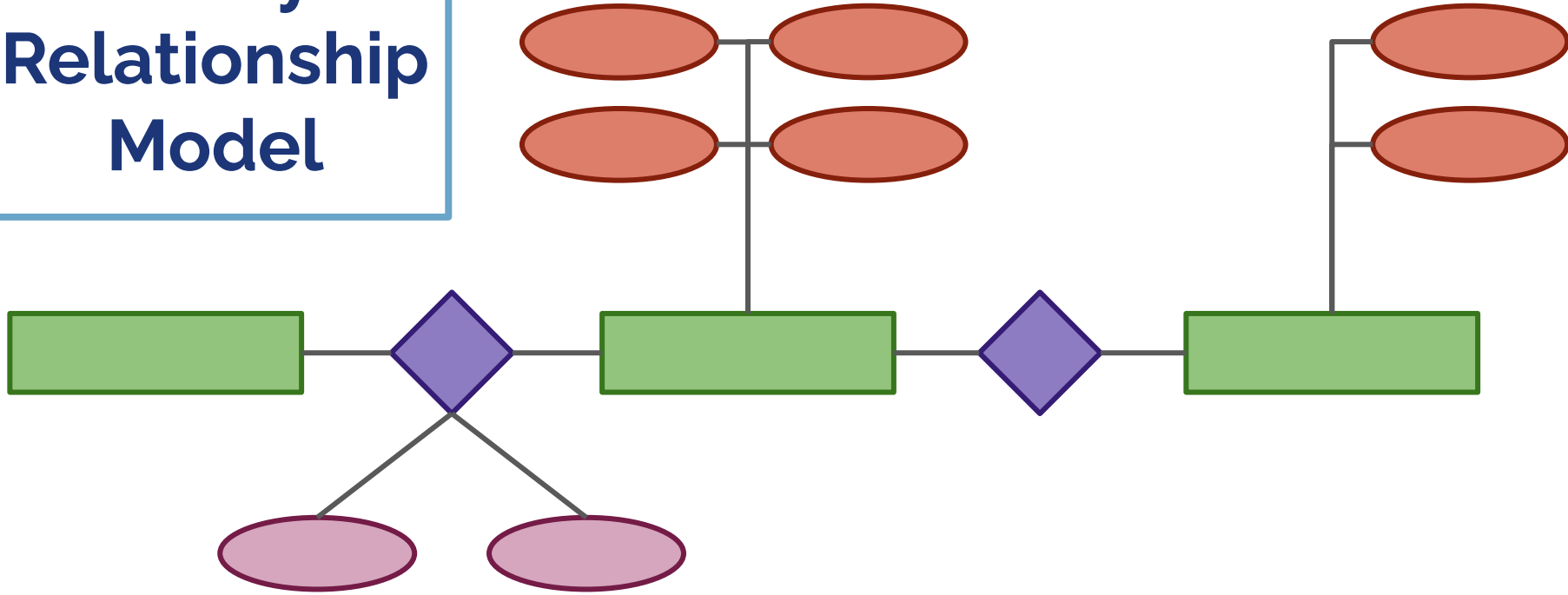
Schemas



Schema - description of the database

Subschema - describes a subset of the database and which users have access to this subset

Entity Relationship Model



Entity-Relationship (ER) Data Model

An **ER Model** is made up of entities, attributes, and the relationships defined between entities.

Entity

An object in the real world that is distinguishable from other objects.

Ex: Employees, Places

Attribute

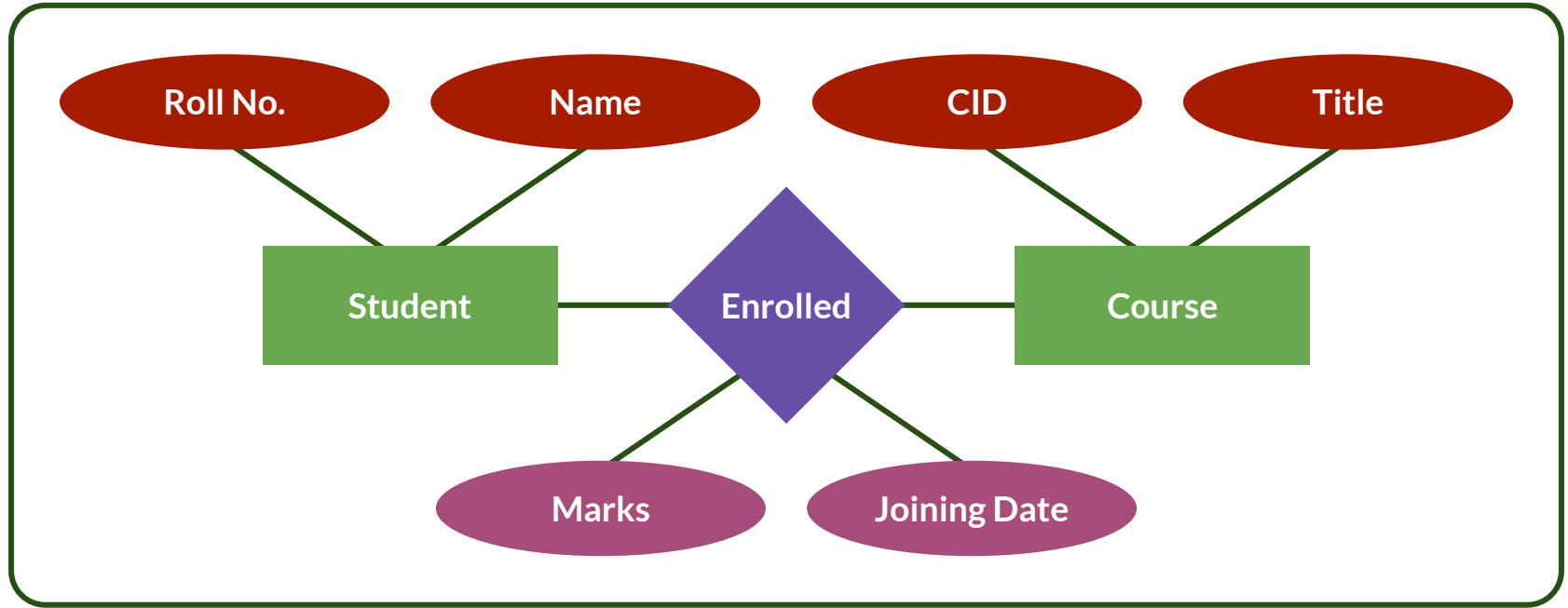
An entity is described in a database based on a set of attributes.

Ex: Name, Age, Gender

Relationship

A relationship links two entities that share one or more attributes.

Ex: Person lives in a House



Entities: Student, Course

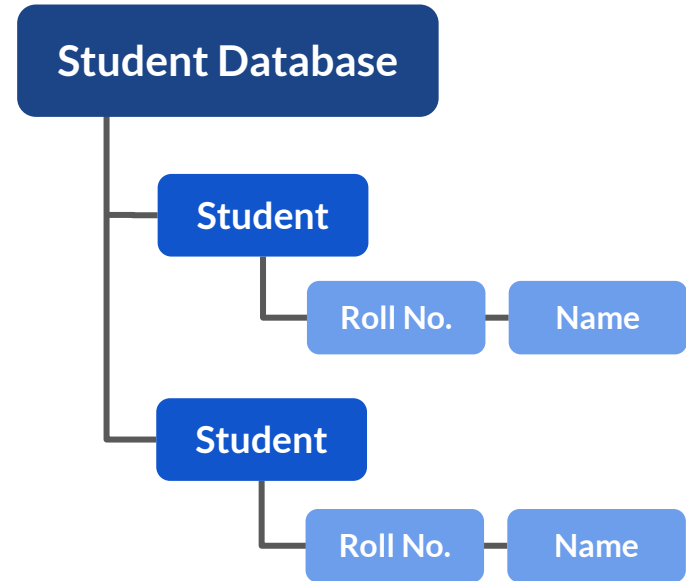
Attributes: Roll No, Name, CID, Title, Marks, Joining Date

Relationship: Enrolled

Defining Data With the ER Model

Instance (Record, Tuple) – single, specific occurrence of an entity

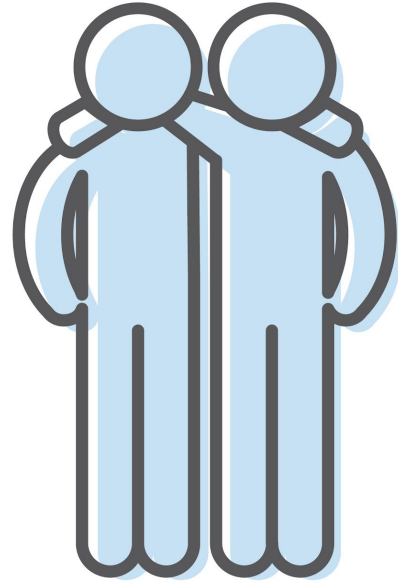
- You can have multiple students and each instance will have one student's RollNo and Name



**Save
Point &
Check In**



Relationships



Relational Model

- Data is viewed as existing in a two dimensional table known as relations
- A relation (table) consists of unique attributes (columns) and tuples (rows)

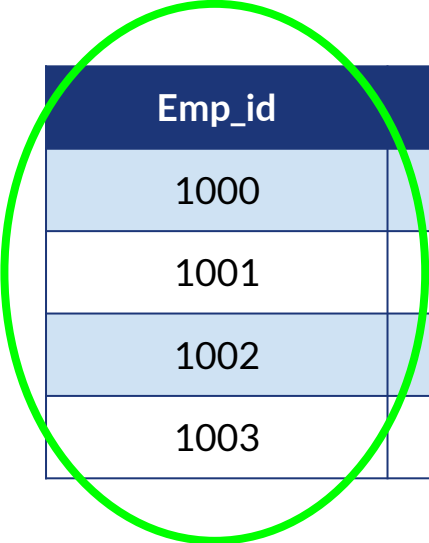
Attributes/Fields/Columns

Emp_id	Emp_name	Emp_age	Emp_email
1000	Derek	24	dk@cognixia.com
1001	Maria	23	ma@cognixia.com
1002	Luis	25	ls@cognixia.com
1003	Janel	25	jl@cognixia.com

Rows/
Records/
Tuples

Keys in Relational Models

Primary Key - uniquely identifies each record in a database table



Emp_id	Emp_name	Emp_age	Emp_email
1000	Derek	24	dk@cognixia.com
1001	Maria	23	ma@cognixia.com
1002	Luis	25	ls@cognixia.com
1003	Janel	25	jl@cognixia.com

Keys in Relational Models

Foreign Key - an attribute in a table that references the primary key of another table

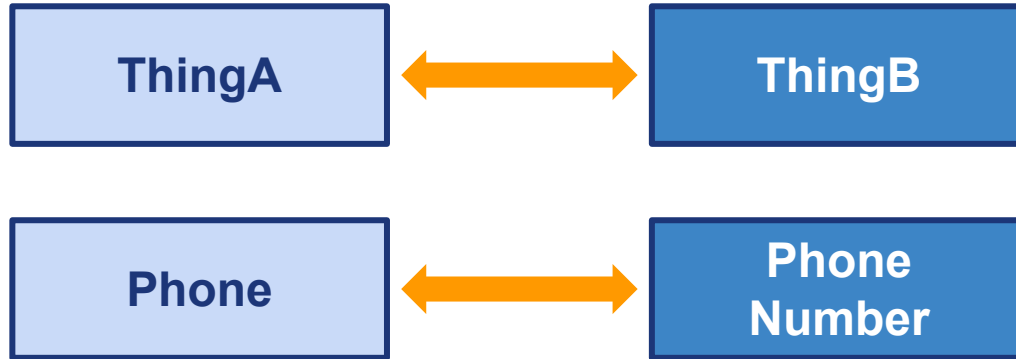
Emp_id	Emp_name	Emp_age	Emp_email	Fk_dept_id
1000	Derek	24	dk@cognixia.com	1
1001	Maria	23	ma@cognixia.com	3
1002	Luis	25	ls@cognixia.com	4
1003	Janel	25	jl@cognixia.com	3

Pk_dept_id	Dept_name
1	Accounting
2	Payroll
3	Marketing
4	HR



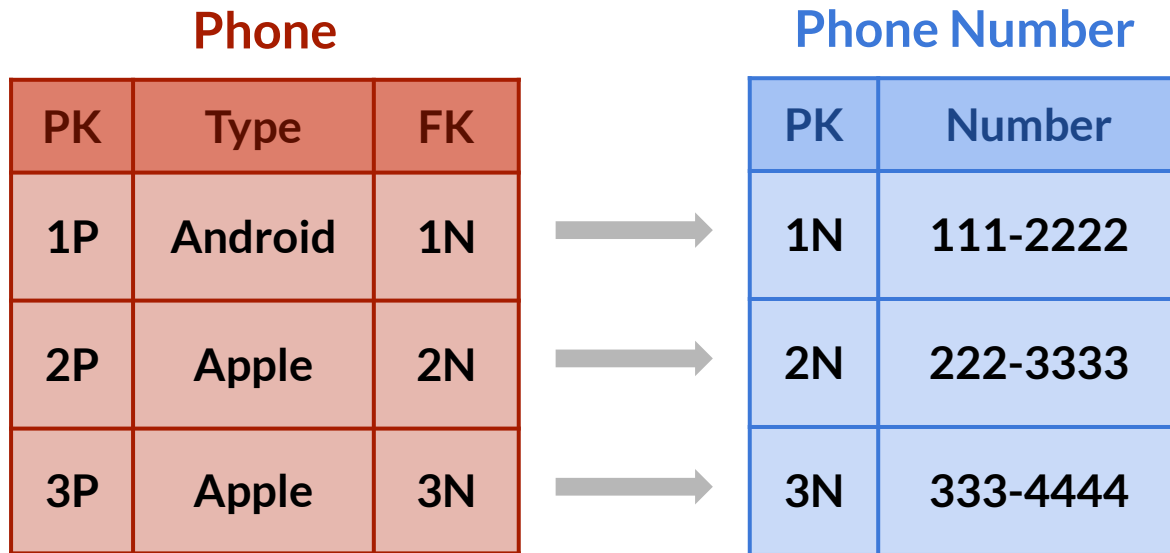
One-to-One

ThingA cannot be related to more than one *ThingB*
ThingB cannot be related to more than one *ThingA*



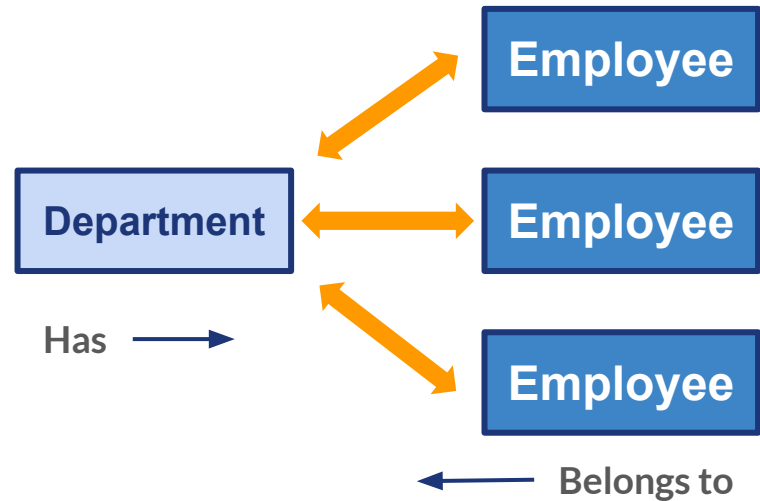
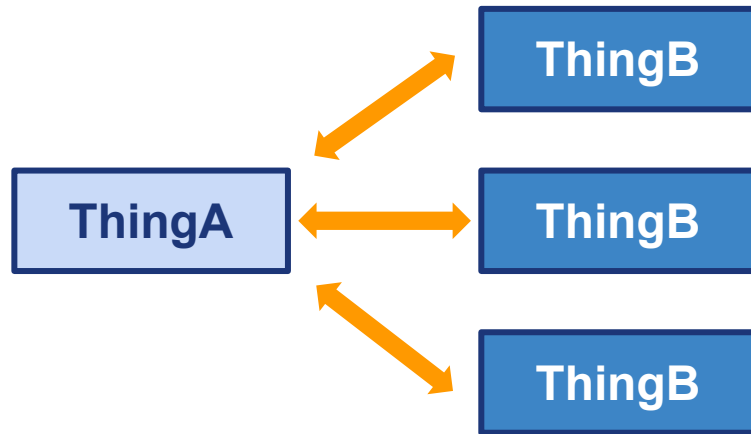
In a database, one table will contain a foreign key column for the second table. The **FK** column in the Phone table will contain a **FOREIGN KEY** constraint and a **UNIQUE** constraint.

Without the unique constraint, cannot enforce a one to one relationship.



One-to-Many

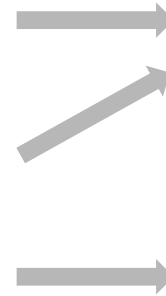
ThingA can be related to more than one *ThingB*
ThingB cannot be related to more than one *ThingA*



The table that will contain the foreign key column is the “many” in the one-to-many relationship. The **FK** column in the Employee only needs the **FOREIGN KEY** constraint to link to the Department table.

Employee

PK	Name	FK
1E	Jonathan	1D
2E	Joseph	1D
3E	Jotaro	3D

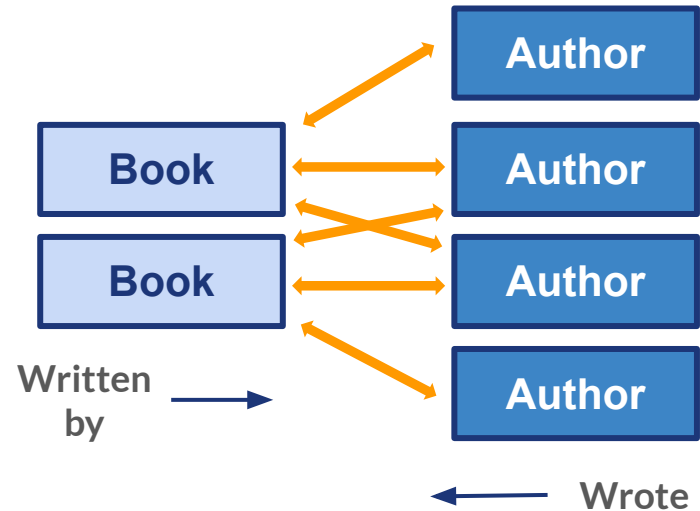
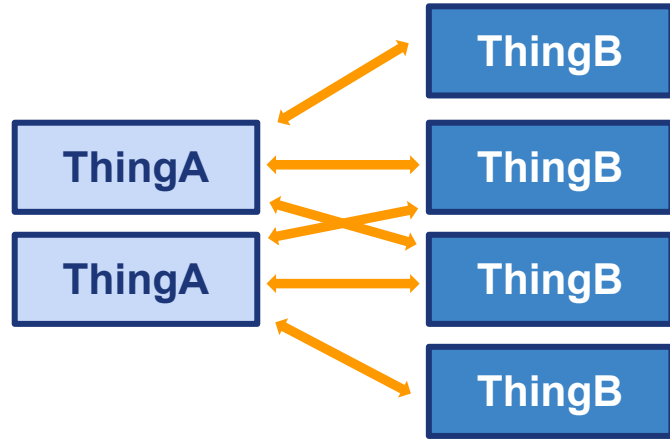


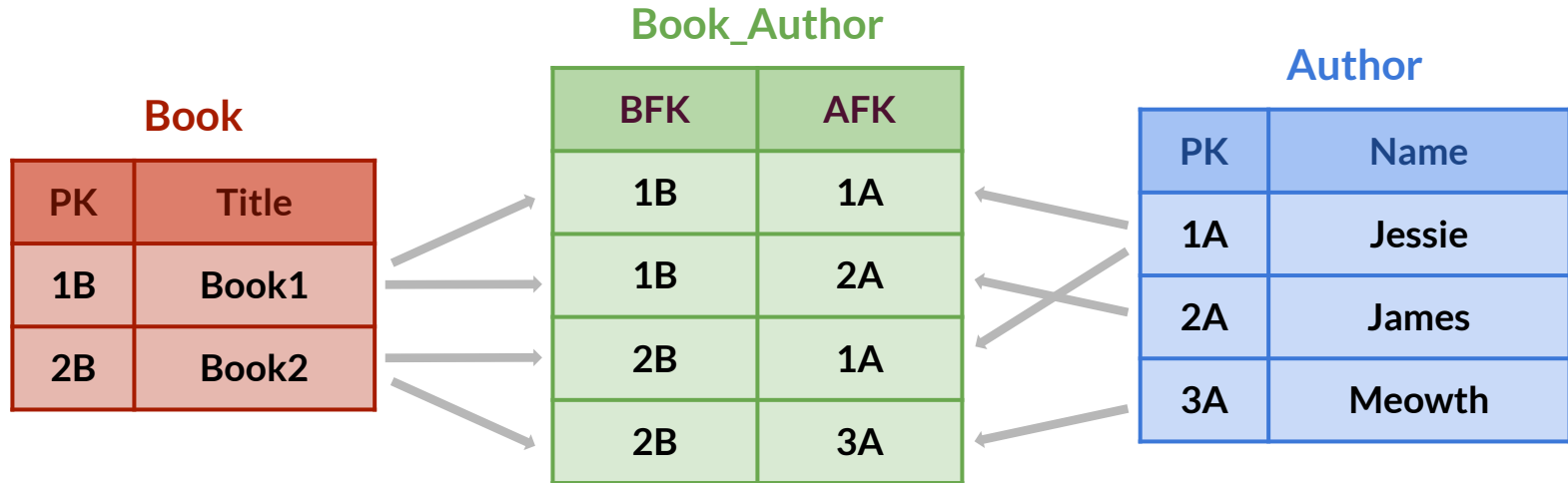
Department

PK	Name
1D	Marketing
2D	Sales
3D	Finance

Many-to-Many

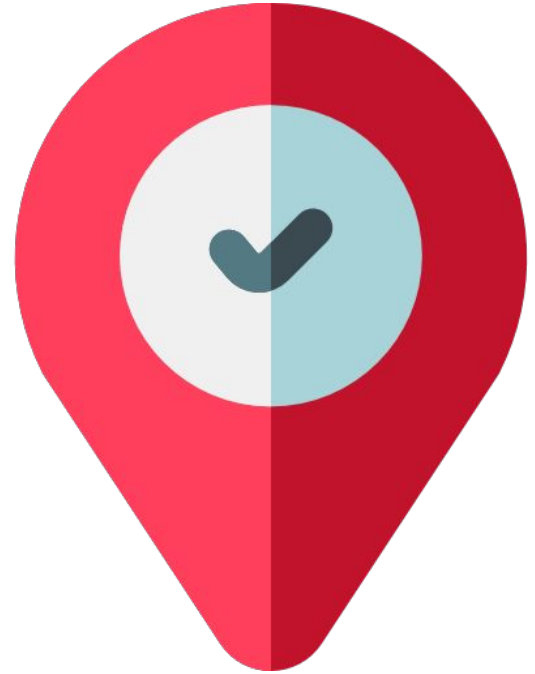
ThingA can be related to more than one *ThingB*
ThingB can be related to more than one *ThingA*





In order to create a many-to-many relationship, an extra table is needed. This **relationship or joining table** will have a foreign key column for each table in this relationship.

**Save
Point &
Check In**

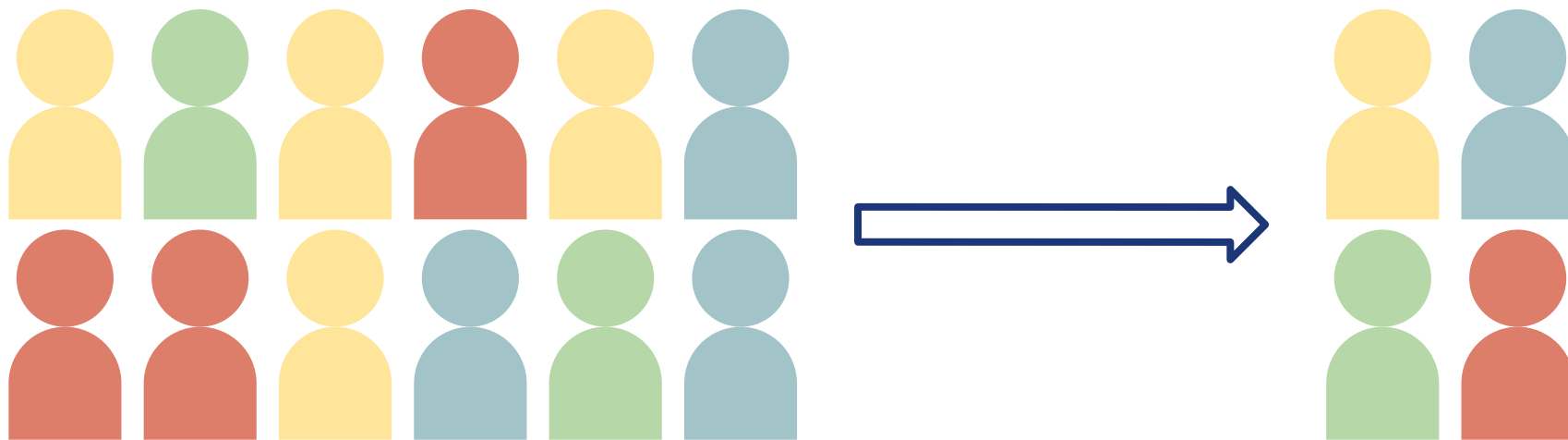


KAHOOT!

1. CLOSE NOTES, USE PHONE OR LAPTOP
2. GO TO KAHOOT.COM
3. CLICK ON ENTER GAME PIN



Normalization



Normalization

Normalization – method for organizing data into a database to eliminate data repetition and undesirable characteristics like

- *Insertion anomalies*
- *Delete anomalies*
- *Update anomalies*



Problems Without Normalization

What are the issues we can run into with the table below?

rollNo	name	department	hod	office_tel
1000	Daisy	CS	Mr. X	53337
1001	Nick	CS	Mr. X	53337
1002	Ana	CS	Mr. X	53337
1003	James	CS	Mr. X	53337

Normalization Example: Un-Normalized

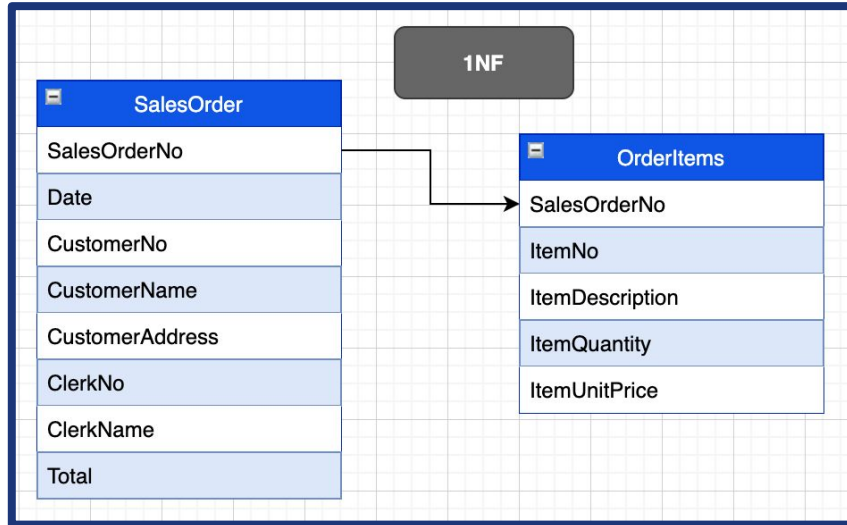
SalesOrders		
SalesOrderNo	ClerkName	Item2UnitPrice
Date	Item1Description	Item3Description
CustomerNo	Item1Quantity	Item3Quantity
CustomerName	Item1UnitPrice	Item3UnitPrice
CustomerAddress	Item2Description	Total
ClerkNo	Item2Quantity	

Normalization Into First Normal Form

- **Separate repeating groups** into new tables
- Start a new table for repeating data
- The **primary key** for the repeating group is usually a **composite key**



Normalization Example: 1NF



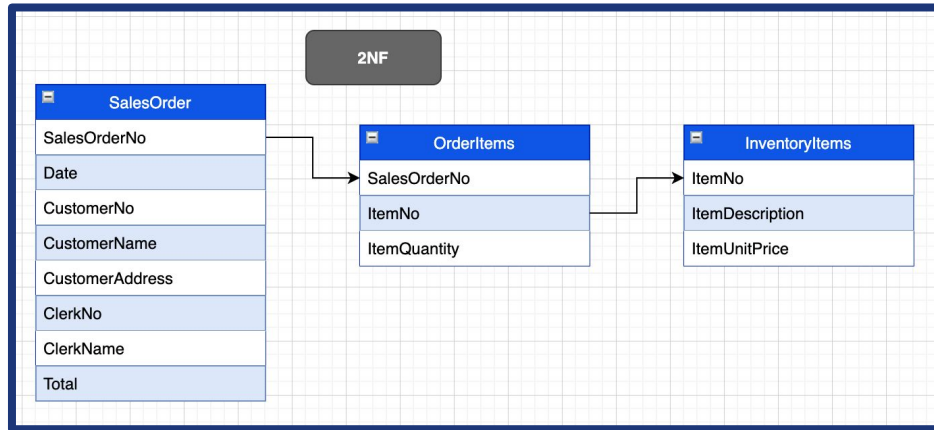
- **OrderItems** table created to separate repeating item information
- Create **ItemNo** and pair it with **SalesOrderNo** as primary key for new table to uniquely identify each item

Normalization Into Second Normal Form

- Remove **partial dependencies** – an attribute depends on only part of the primary key
- Start a new table for partially dependent data and part of key it depends on



Normalization Example: 2NF



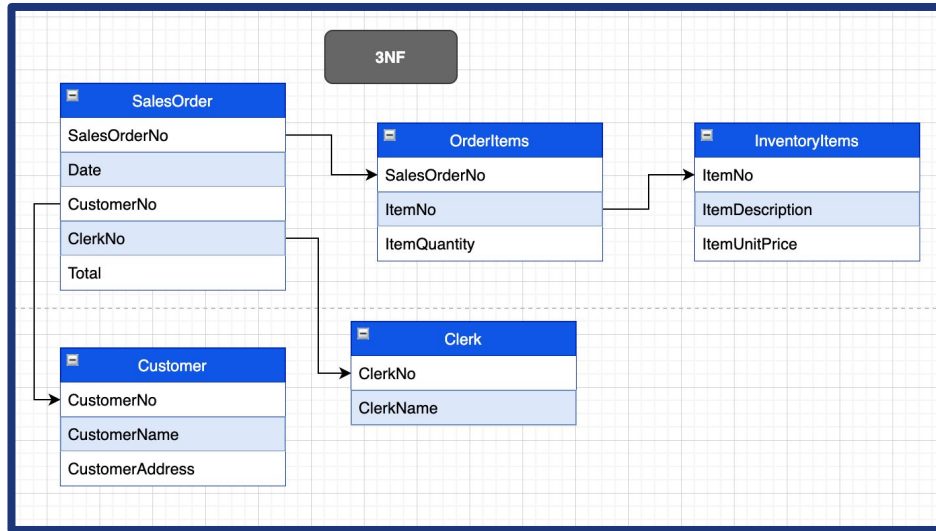
- Removed the partial dependency caused by Item
- Item description not duplicated
- Don't need to insert sales order to add inventory item
- Item information stays even if sales order is deleted
- To change an item description, you don't need to go through each sales order

Normalization Into Third Normal Form

- Remove **transitive dependencies** – attribute depends on an attribute other than the primary key
- Start a new table for transitive dependency and attributes that depend on it
- Keep a copy of the key attribute in the original table



Normalization Example: 3NF



- Removed the transitive dependencies with Customer and Clerk tables
- Now customer and clerk information isn't in every order
- If you delete a sales order, you aren't deleting customer or clerk information
- Don't need to change all orders when you need to update customer or clerk information

**Save
Point &
Check In**



KAHOOT!

1. CLOSE NOTES, USE PHONE OR LAPTOP
2. GO TO KAHOOT.COM
3. CLICK ON ENTER GAME PIN





data.sql

```
SELECT * FROM chef;
```

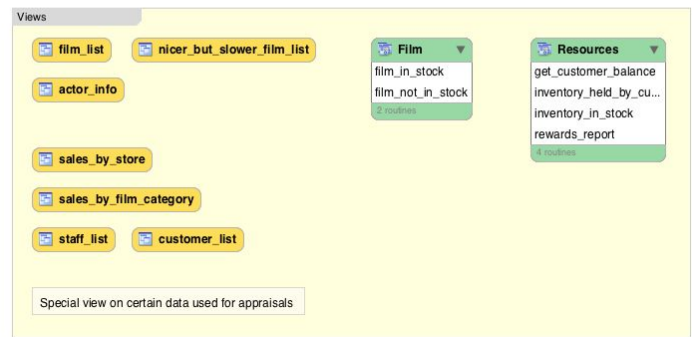
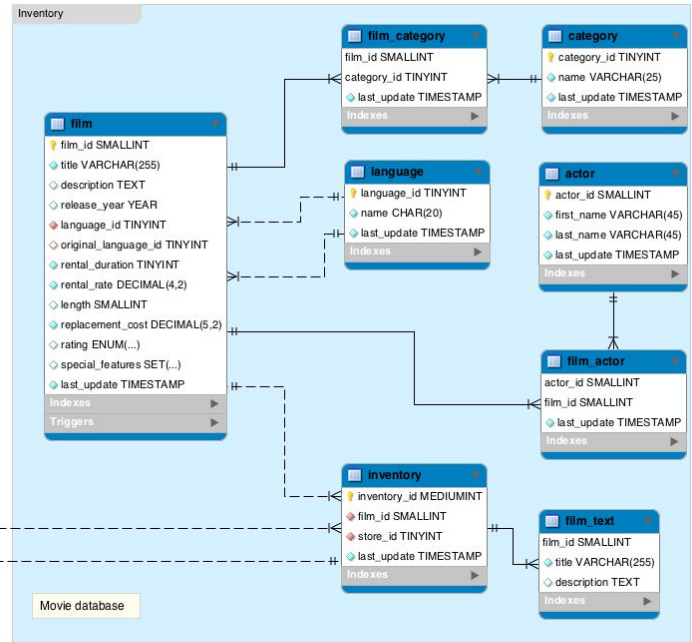
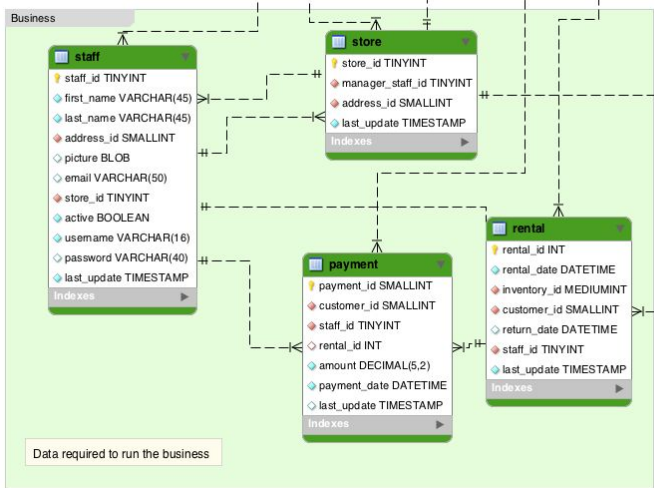
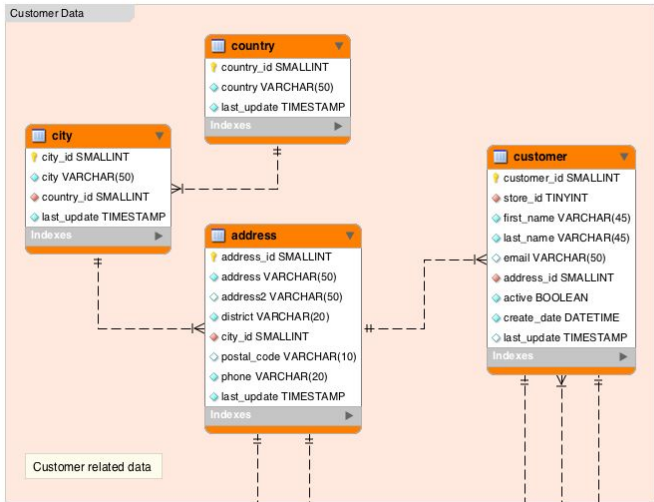
MySQL

chef_id	name	best_dish	rest_id	...
1	Chef Ramsey	Lamb	1	...
2	Chef Remy	Ratatouille	2	...
3	Chef Flay	Roast	1	...
4	Chef Boyardee	Ravioli	3	...
5	Chef Child	Steak	3	...
6	Swedish Chef	Swedish Fööd	2	...



MySQL™

MySQL is an open-source relational database management system. Its name is a combination of "My", the name of co-founder Michael Widenius's daughter, and "SQL", the abbreviation for Structured Query Language.



Sakila Database

Text Data Types

Data Type	Description
CHAR(size)	Fixed length string, stores up to 255 characters
VARCHAR(size)	Variable length string, stores up to 255 characters
TINYTEXT	String with max 255 characters
TEXT	String with max 65,535 characters
BLOB	Binary Large Objects, holds up to 65,535 bytes
MEDIUMTEXT	String with max 16,777,215 characters
MEDIUMBLOB	Binary Large Objects, holds up to 16,777,215 bytes
LONGTEXT	String with max 4,294,967,295 characters
LOBLOB	Binary Large Objects, holds up to 4,294,967,295 bytes

Number Data Types

Data Type	Description
TINYINT(size)	-128 to 127 normal, 0 to 255 UNSIGNED
SMALLINT(size)	-32768 to 32767 normal, 0 to 65535 UNSIGNED
MEDIUMINT(size)	-8388608 to 8388607 normal, 0 to 16777215 UNSIGNED
INT(size)	-2147483648 to 2147483647 normal, 0 to 4294967295 UNSIGNED
BIGINT(size)	-9223372036854775808 to 9223372036854775807 normal, 0 to 18446744073709551615 UNSIGNED
FLOAT(size, d)	Small floating decimal point number, d parameter is max digits to the right of decimal point
DOUBLE(size, d)	Large floating decimal point number, d parameter is max digits to the right of decimal point
DECIMAL(size, d)	A DOUBLE stored as a string, allowing for a fixed decimal point

Date Data Types

Data Type	Description
DATE()	A date. Format: YYYY-MM-DD <u>Range</u> : '1000-01-01' to '9999-12-31'
DATETIME()	Date and time. Format: YYYY-MM-DD HH:MI:SS <u>Range</u> : '1000-01-01 00:00:00' to '9999-12-31 23:59:59'
TIMESTAMP()	A timestamp, values stored as number of seconds since the Unix epoch . Format: YYYY-MM-DD HH:MI:SS <u>Range</u> : '1970-01-01 00:00:01' to '2038-01-09 03:14:07' UTC
TIME()	A time. Format: HH:MI:SS <u>Range</u> : '-838:59:59' to '838:59:59'
YEAR()	A year in a two-digit or four-digit format <u>Two-Digit Range</u> : 70 to 69, represents years 1970-2069 <u>Four-Digit Range</u> : 1901 to 2155

Constraints

Constraints are used to define **rules** that allow/restrict values stored in columns

- **NOT NULL** - none of the values in this column can be null
- **UNIQUE** - each value must be unique, no repeating values
- **PRIMARY KEY** - set column as primary key
- **FOREIGN KEY** - set column as foreign key
- **DEFAULT** - if no value given, set default value for column
- **CHECK** - allows values to be inserted if they pass boolean check (like a custom constraint)





ddl.sql

Data Definition Language

DDL is used to define the database structure or schema

- **CREATE**
- **ALTER**
- **DROP**
- **TRUNCATE**

CREATE

- **CREATE** is used to create databases, tables, and other structures
- Run keyword **USE** to indicate where you are creating a table

ddl.sql

```
CREATE DATABASE eateries;
USE eateries;
CREATE TABLE restaurant (
    rest_id int auto_increment,
    ...
    PRIMARY KEY (rest_id)
);
CREATE TABLE chef (
    chef_id int PRIMARY KEY,
    name varchar(30) NOT NULL,
    rest_id int NOT NULL,
    FOREIGN KEY (rest_id) references
        restaurant(rest_id)
);
```

ALTER

- **ALTER** is used to change/alter structures like tables
- You can alter a table by
 - ◆ Adding a column
 - ◆ Modifying a column
 - ◆ Deleting a column

ddl.sql

```
# add a new column for the best_dish
# with varchar(100)
ALTER TABLE chef ADD COLUMN best_dish
    varchar(100);

# modify best_dish, reduce size to 50
# and add a default value
ALTER TABLE chef MODIFY COLUMN best_dish
    varchar(50) DEFAULT 'steak';

# delete the column best_dish
ALTER TABLE chef DROP best_dish;
```

DROP

- **DROP** is used to delete a structure
- Double check what you are dropping, once deleted, it cannot be recovered

ddl.sql

```
# double check you really want to delete  
# this table
```

```
DROP TABLE chef;
```

```
# cannot drop restaurant before chef  
# because chef is dependant on it
```

```
DROP TABLE restaurant;
```

TRUNCATE

- **TRUNCATE** will delete all the records in a table without deleting the table structure
- If using **DROP**, table data and structure would be removed

ddl.sql

```
# just removes records
TRUNCATE TABLE chef;

# will show an empty table
SELECT * FROM chef;

...
```




ddl.sql

dml.sql

Data Manipulation Language

DML is used for managing data within schema objects

- **INSERT**
- **UPDATE**
- **DELETE**
- **SELECT**

INSERT

- **INSERT** is used to add a record to a table
- There are two main ways to write insert
 - ◆ Without specifying columns
 - ◆ Specifying columns

ddl.sql dml.sql

```
# inserts values for Chef Ramsey
INSERT INTO chef
    VALUES (null, 'Chef Ramsey', 'lamb',
            1);

# insert specifying columns
INSERT INTO chef(chef_id, name,
best_dish, rest_id)
    VALUES (null, 'Chef Ramsey', 'lamb',
            1);

# insert without all columns listed
INSERT INTO chef(name, rest_id)
    VALUES ('Chef Ramsey', 1);
```