Week 7

Database Design

The Design Process[1]

1. **Determine the purpose of your database**
   - What is the high level purpose of this database?
2. **Find and organize the information required**
   - What information should be stored to fulfill the high level purpose?
3. **Divide the information into tables**
   - What are the entities to store?
4. **Turn information items into columns**
   - What are the individual data pieces that make up those entities (tables)?
5. **Specify primary keys**
   - What is the unique field (column) for each entity (table)?
6. **Set up the table relationships (and constraints)**
   - Do these tables relate? If so what is the connection?
7. **Refine your design**
   - Look for errors to inform improvements.
8. **Apply the normalization rules**
   - Review normalization rules and adjust as necessary.

[1] Microsoft - DB Design Basics - The design process

Relationships Between Tables

- With tables, columns, and unique ids decided on… tables can be connected!
- Goals:
  a. Identify where there are **relationships** between tables
  b. Identify the relationship **type** between those tables

Finding Relationships

- Look for matching data in fields
  - This is often a field with the same name or data in different tables

instructor

name	birth_date
Matthew Bebis	1912-06-23

course

instructor	code	section	term
Matthew Bebis	HTTP5126	A	F24
Matthew Bebis	HTTP5126	B	F24
Matthew Bebis	HTTP5122	A	S24

- Determine if a table **HAS A** entity that is represented by another table
  - Example above: a course **HAS AN** instructor OR an instructor **HAS A** course to teach

Relationship Types

- When a relationship is found, it will have a relationship type
- **Relationships** will fall into 1 of 3 categories
  a. one-to-one relationship
  b. one-to-many relationship
  c. many-to-many relationship

one-to-one relationship

country

name	population	capital_city
Canada	38.93 million	Ottawa
Denmark	5.9 million	Copenhagen

capital_city

city	population
Ottawa	1 million
Copenhagen	600 k

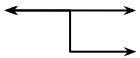
- 1 row in a table connects to exactly 1 row in a different table
  - Example above: A country **HAS A** capital city

Relationship Types

one-to-many relationship

country

name	population
Canada	38.93 million
Denmark	5.9 million



city

name	city	population
Canada	Toronto	3 million
Canada	Vancouver	700 k

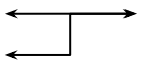
- 1 row in a table connects to many different rows in a different table

  OR
- Many rows in a table connect to 1 row in a different table
  - Example above: A country **HAS MANY** cities, a city **HAS A** country

many-to-one relationship *same as one-to-many

name	city	population
Canada	Toronto	3 million
Canada	Vancouver	700 k

city



name	population
Canada	38.93 million
Denmark	5.9 million

country

Relationship Types

one-to-many relationship / Example

name	population
Canada	38.93 million
Denmark	5.9 million

name	city	population
Canada	Toronto	3 million
Canada	Vancouver	700 k
Canada	Ottawa	1 million
Denmark	Copenhagen	600 k
Denmark	Odense	180 k

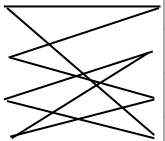
- Each **ONE** country **HAS MANY** cities
  - Canada has many cities
  - Denmark has many cities
- The **MANY** cities can **HAVE** just **ONE** country
  - Toronto, Vancouver, and Ottawa all belong to Canada
  - Copenhagen and Odense all belong to Denmark

Relationship Types

many-to-many relationship

student

number	name
n01234567	Auston Matthews
n99999999	Vladimir Guerrero Jr.
n00000000	Natalie Spooner
n98765432	RJ Barrett



course

instructor	code	section	term	class_list
Matthew Bebis	HTTP5126	A	F24	Auston Matthews, Vladimir Guerrero Jr.
Matthew Bebis	HTTP5126	B	F24	Natalie Spooner, RJ Barrett
Sean Doyle	HTTP5122	A	F24	Vladimir Guerrero Jr., RJ Barrett
Sean Doyle	HTTP5122	B	F24	Natalie Spooner, Auston Matthews

- many rows in a table connect to many rows in a different table
  - Example:
  - A student **HAS MANY** courses
  - A course **HAS MANY** students

Relationship Types / Bridge Tables[1]

*listing items in 1 cell indicates need for another table, connected with either a one-to-many or many-to-many relationship

many-to-many relationship

student

number	name
n01234567	Auston Matthews
n99999999	Vladimir Guerrero Jr.
n00000000	Natalie Spooner
n98765432	RJ Barrett

course

instructor	code	section	term	class_list
Matthew Bebis	HTTP5126	A	F24	Auston Matthews, Vladimir Guerrero Jr.
Matthew Bebis	HTTP5126	B	F24	Natalie Spooner, RJ Barrett
Sean Doyle	HTTP5122	A	F24	Vladimir Guerrero Jr., RJ Barrett
Sean Doyle	HTTP5122	B	F24	Natalie Spooner, Auston Matthews

student_number	course_id
n01234567	HTTP5126-A-F24
n99999999	HTTP5126-A-F24
n00000000	HTTP5126-B-F24
n01234567	HTTP5126-B-F24
n99999999	HTTP5122-A-F24
n98765432	HTTP5122-A-F24

- many-to-many relationships require a **bridging table** to connect them
- Each record in one table can relate to multiple records in another table, and vice versa.
- With this, neither table directly relates to the other, instead both relate through the bridge

[1] Bridge Table in Database

Keys - Specify Primary Keys

- Every table should include a column that uniquely identifies each row
- The unique column is the **primary key** for the table
  - Primary key is referenced *(by FK*)* in **creating** the **links** (relationships) **between tables**
  - The primary key must be different in every row, there **cannot be duplicated values**
  - Primary keys **should not change** their value
- Primary Keys can be any column in your table, as long as they are unique
  i. The key can be a column of data that is unique
  ii. The key can also be a combination of columns that when combined make a unique value
  iii. Typically creating an “id” column as the unique identifier is easiest

i. **Primary Key**

number	name
n01234567	Auston Matthews

ii. **(Primary) Composite Key**

code	instructor
HTTP5126-A	Matthew Bebis

iii. **Primary Key**

bus_id	route
1	508

Keys - Specify Foreign Keys[1]

- **Foreign Key:** a column(s) in one table **that refers to the Primary Key** in another table, to formalize the relationship between tables
- Foreign Keys can reference any type of primary key from other tables
  - i. Unique Data PK*
  - ii. Composite PK*
  - iii. id PK*

i.

PK

number	name
n01234567	Auston Matthews

Foreign Key

number	address
n01234567	40 Bay St.

ii.

(Primary) Composite Key

code	instructor
HTTP5126	Matthew Bebis

(Foreign) Composite Key

code	instructor
HTTP5126	Matthew Bebis

iii.

PK

bus_id	route
1	508

FK

bus_id	driver
1	Sean Doyle

[1] What are Foreign Keys?

Keys - Specify Foreign Keys

one-to-one relationship

country

name *(PK)*	population	capital_city *(FK)*
Canada	38.93 million	Ottawa
Denmark	5.9 million	Copenhagen

capital city

name *(PK)*	population
Ottawa	1 million
Copenhagen	600 k

one-to-many relationship

country

name *(PK)*	population
Canada	38.93 million
Denmark	5.9 million

city

country *(FK)*	name	population
Canada	Toronto	3 million
Canada	Vancouver	700 k

many-to-many relationship

Composite PK **That also acts as a FK*

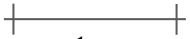
number *(PK)*	name
n01234567	Auston Matthews
n99999999	Vladimir Guerrero Jr.
n00000000	Natalie Spooner
n98765432	RJ Barrett

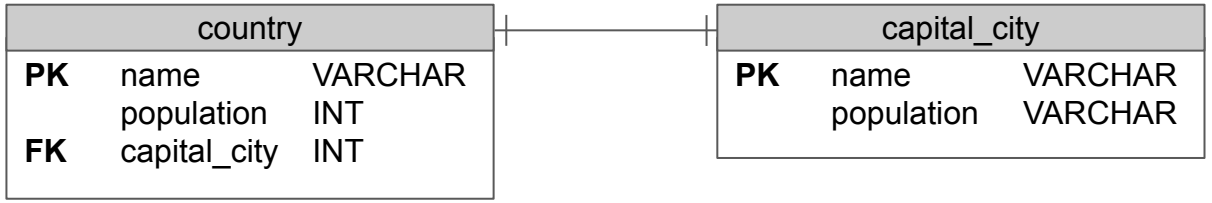
student_number	course_id
n01234567	1
n99999999	1
n00000000	2
n01234567	2

course_id *(PK)*	instructor	code	section	term
1	Matthew Bebis	HTTP5126	A	F24
2	Matthew Bebis	HTTP5126	B	F24
3	Sean Doyle	HTTP5122	A	F24
4	Sean Doyle	HTTP5122	B	F24

Entity Relationship Diagrams (ERD) *Evolution of Table Diagrams

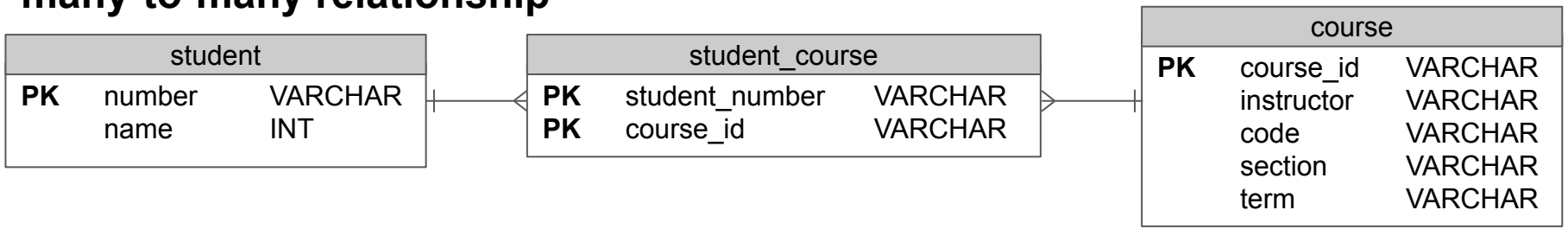
one-to-one relationship



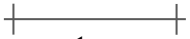


one-to-many relationship

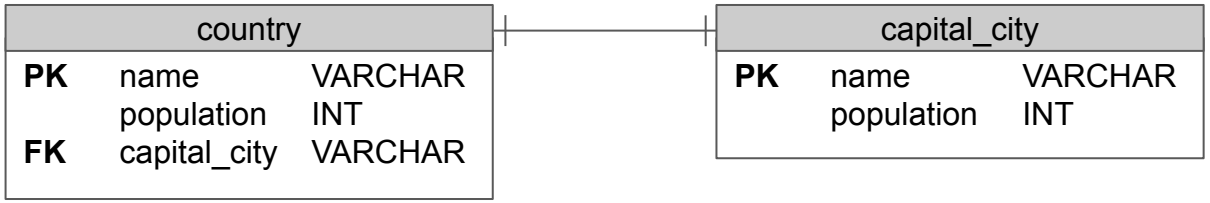
many-to-many relationship



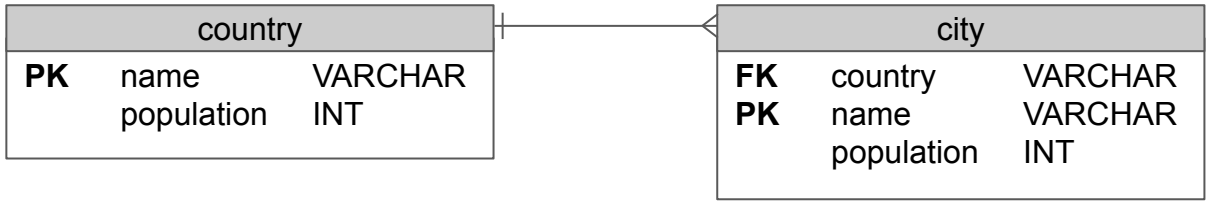
Multiple Relationship Types



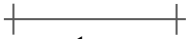
one-to-one relationship



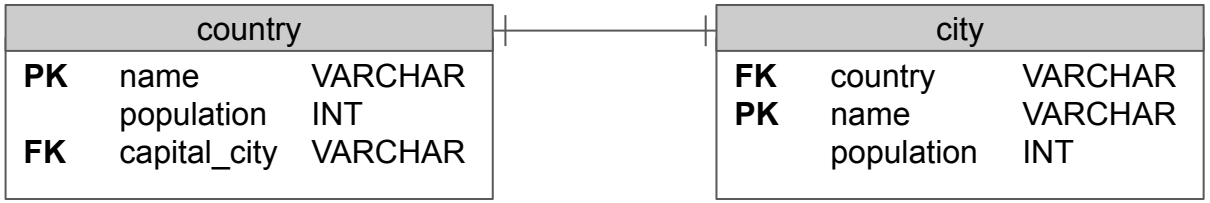
one-to-many relationship



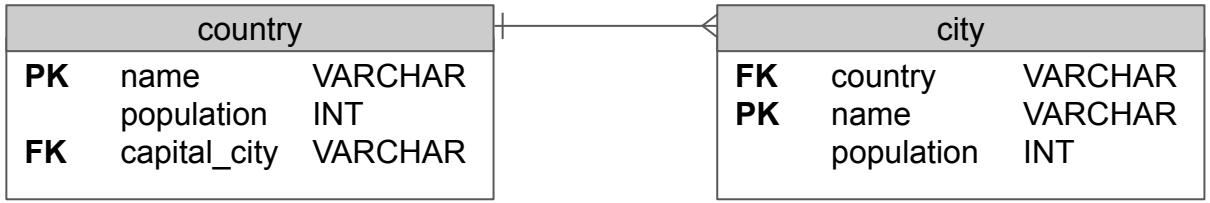
Multiple Relationship Types



one-to-one relationship



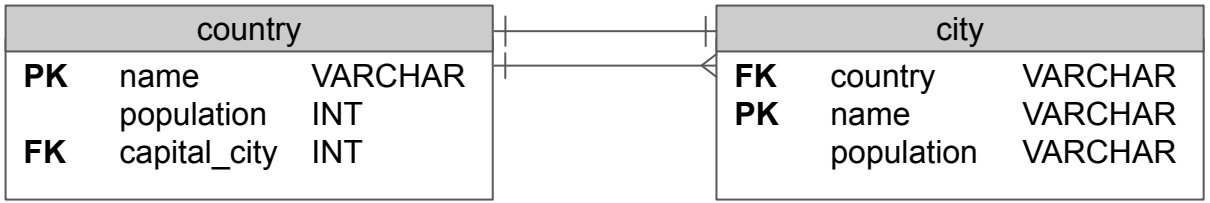
one-to-many relationship



Multiple Relationship Types

one-to-one relationship *AND*

one-to-many relationship



- It is ok for 2 tables create multiple relationships with each other, as long as the tables pass the normalization rules

Table Constraints

- Constraints are used to specify rules and limit the data for columns in our tables
- This ensures the accuracy and reliability of the data in the table
- Violations of constraints abort queries
- Constraints can be column level or table level

Constraint Types[1]

NOT NULL - Ensures that a column cannot have a NULL value

UNIQUE - Ensures that all values in a column are different

*__PRIMARY__ and **FOREIGN KEYS** are also constraints

PRIMARY KEY - Both **NOT NULL** & **UNIQUE**

[1] SQL Constraints

NOT NULL[1]

instructor		
PK	instructor_id	INT
N	name	VARCHAR

UNIQUE[2]

instructor		
PK	instructor_id	INT
N	name	VARCHAR
U	employee_id	VARCHAR

NOT NULL & UNIQUE

instructor		
PK	instructor_id	INT
N	name	VARCHAR
U/N	employee_id	VARCHAR

[1] SQL Constraint NOT NULL
[2] SQL Constraint UNIQUE

Entity Relationship Diagrams (ERD)

- Used to represent a single databases architecture
- ERDs have varying layouts but will generally show this information:
  - Entities (tables)
  - Fields (columns)
  - Data types
  - Relationships
  - Constraints

Refine your design with Guiding Principles[1]

- **Reduce Redundancy,** duplicate information is bad
  - It wastes space and increases the likelihood of errors and inconsistencies
- **Correctness and Completeness**, no exceptions
  - Incorrect data leads to incorrect data driven decisions

A good database design is, therefore, one that:

- Divides your information into subject-based tables to reduce redundant data
- Creates relationships between tables to join information together as needed
- Helps support and ensure the accuracy and integrity of your information
- Accommodates data processing and reporting needs

[1] Microsoft - DB Design Basics - What is good database design?

Normalization Forms

Normalization is the process of organizing data in a database to eliminate redundancy and inconsistencies

Normalization can be achieved by applying each normal form step by step from the earliest normal forms to the latest

The normal forms are:

1NF - **1**st **N**ormal **F**orm

2NF - **2**nd **N**ormal **F**orm

3NF - **3**rd **N**ormal **F**orm

4NF - **4**th **N**ormal **F**orm

ETNF - **E**ssential **T**uple **N**ormal **F**orm

5NF - **5**th **N**ormal **F**orm

DKNF - **D**omain-**K**ey **N**ormal **F**orm

6NF - **6**th **N**ormal **F**orm

Normalized DBs

- There are differing opinions for when a database is considered normalized
- Some argue normalization occurs after 5NF
- It is best to assume a DB is normalized when it meets the standards of your use case
- 3NF is is generally accepted as normalized

Normal Forms

First Normal Form (1NF)

- Each attribute (column) has a unique name
- Domain of attributes must not change (same data type)
- Each row is uniquely identifiable (no duplicate rows)
- Each cell must have only a single value (atomicity)

Second Normal Form (2NF) – Including 1NF

- Every column is dependent on the whole primary key

Third Normal Form (3NF) – Including 1NF, 2NF

- All non-key columns are functionally dependent on the primary key

Normal Forms

Example Unnormalized Order Table

customer_name	product_name	supplier	supplier_location	price	brand	order_id	email
John	Laptop	XYZ Electronics	Toronto	800	Dell	1	john@gmail.com
Jane	Smartphone	ABC Gadgets	Montreal	Sa:600	Apple, Samsung	2	jane@gmail.com
John	Camera	XYZ Electronics	Toronto	300	Canon	3	john@gmail.com
Alice	Chair	XYZ Furniture	Vancouver	100	Herman Miller	4	alice@gmail.com

Apply 1st Normal Form

customer_name	product_name	supplier	supplier_location	price	brand	order_id	email
John	Laptop	XYZ Electronics	Toronto	800	Dell	1	john@gmail.com
Jane	Smartphone	ABC Gadgets	Montreal	Sa:600	Apple, Samsung	2	jane@gmail.com
John	Camera	XYZ Electronics	Toronto	300	Canon	3	john@gmail.com
Alice	Chair	XYZ Furniture	Vancouver	100	Herman Miller	4	alice@gmail.com

1st Normal Form

✔ Each column has a unique name

✖ Every cell in a column must hold the same data type

✓ Each row is uniquely identifiable (no duplicate rows)

✖ A cell may not contain a set of values or a nested record

Apply 1st Normal Form

Candidate primary key

customer_name	supplier	supplier_location	order_id	email	product_id
John	XYZ Electronics	Toronto	1	john@gmail.com	1
Jane	ABC Gadgets	Montreal	2	jane@gmail.com	2
Jane	ABC Gadgets	Montreal	2	jane@gmail.com	3
John	XYZ Electronics	Toronto	3	john@gmail.com	4
Alice	XYZ Furniture	Vancouver	4	alice@gmail.com	5

product_id	name	price	brand
1	Laptop	800	Dell
2	Smartphone	600	Apple
3	Smartphone	600	Samsung
4	Camera	300	Canon
5	Chair	100	Herman Miller

1st Normal Form

- ✔ Each column has a unique name
- ✔ Every cell in a column must hold the same data type
- ✔ Each row is uniquely identifiable (no duplicate rows)
- ✔ A cell may not contain a set of values or a nested record
  - If it does, it should be extracted to its own table and relate back with a relationship

Apply 2nd Normal Form

Candidate primary key

customer_name	supplier	supplier_location	order_id	email	product_id
John	XYZ Electronics	Toronto	1	john@gmail.com	1
Jane	ABC Gadgets	Montreal	2	jane@gmail.com	2
Jane	ABC Gadgets	Montreal	2	jane@gmail.com	3
John	XYZ Electronics	Toronto	3	john@gmail.com	4
Alice	XYZ Furniture	Vancouver	4	alice@gmail.com	5

2nd Normal Form

- ✓ 1NF
- ✖ Every column must depend on the **whole primary key**, not just part of it
  - Fields that do not depend on the primary key should be extracted to a new or existing table in which they do depend on the primary key

Apply 2nd Normal Form

customer_name	order_id	email	product_id
John	1	john@gmail.com	1
Jane	2	jane@gmail.com	2
Jane	2	jane@gmail.com	3
John	3	john@gmail.com	4
Alice	4	alice@gmail.com	5

supplier	supplier_location
XYZ Electronics	Toronto
ABC Gadgets	Montreal
ABC Gadgets	Montreal
XYZ Electronics	Toronto
XYZ Furniture	Vancouver

No primary key

2nd Normal Form

- ✖ 1NF - Each row is uniquely identifiable (no duplicate rows)
- ✖ Every column must depend on the **whole primary key**, not just part of it
  - Fields that do not depend on the primary key should be extracted to a new or existing table in which they do depend on the primary key

Apply 2nd Normal Form

customer_name	order_id	email	product_id
John	1	john@gmail.com	1
Jane	2	jane@gmail.com	2
Jane	2	jane@gmail.com	3
John	3	john@gmail.com	4
Alice	4	alice@gmail.com	5

supplier_id (PK)	name	location
1	XYZ Electronics	Toronto
2	ABC Gadgets	Montreal
3	XYZ Furniture	Vancouver

product_id	name	price	brand	supplier_id
1	Laptop	800	Dell	1
2	Smartphone	600	Apple	2
3	Smartphone	600	Samsung	2
4	Camera	300	Canon	1
5	Chair	100	Herman Miller	3

2nd Normal Form

- ✓ 1NF
- ✓ Every column must depend on the **whole primary key**, not just part of it
  - Fields that do not depend on the primary key should be extracted to a new or existing table in which they do depend on the primary key

Apply 3rd Normal Form

customer_name	order_id	email	product_id
John	1	john@gmail.com	1
Jane	2	jane@gmail.com	2
Jane	2	jane@gmail.com	3
John	3	john@gmail.com	4
Alice	4	alice@gmail.com	5

*Customer name depends on email

3rd Normal Form

✓ 2NF

✖ Columns should not depend on non primary keys

- If a column depends on another column that is not the primary key, it will violate 3NF
- Columns that depend on non primary key columns should be extracted to their own table using the depended upon column as a primary key

Apply 3rd Normal Form

order_id	email	product_id
1	john@gmail.com	1
2	jane@gmail.com	2
2	jane@gmail.com	3
3	john@gmail.com	4
4	alice@gmail.com	5

email	customer_name
john@gmail.com	John
jane@gmail.com	Jane
jane@gmail.com	Jane
john@gmail.com	John
alice@gmail.com	Alice

3rd Normal Form

✖ 2NF - 1NF - Each row is uniquely identifiable (no duplicate rows)

✓ Columns should not depend on non primary keys

- If a column depends on another column that is not the primary key, it will violate 3NF
- Columns that depend on non primary key columns should be extracted to their own table using the depended upon column as a primary key

Apply 3rd Normal Form

order_id	email	product_id
1	john@gmail.com	1
2	jane@gmail.com	2
2	jane@gmail.com	3
3	john@gmail.com	4
4	alice@gmail.com	5

email	customer_name
john@gmail.com	John
jane@gmail.com	Jane
alice@gmail.com	Alice

3rd Normal Form

- ✓ 2NF - 1NF - Each row is uniquely identifiable (no duplicate rows)
- ✓ Columns should not depend on non primary keys
  - If a column depends on another column that is not the primary key, it will violate 3NF
  - Columns that depend on non primary key columns should be extracted to their own table using the depended upon column as a primary key

Apply 3rd Normal Form

Candidate primary key

order_id	email	product_id
1	john@gmail.com	1
2	jane@gmail.com	2
2	jane@gmail.com	3
3	john@gmail.com	4
4	alice@gmail.com	5

email	customer_name
john@gmail.com	John
jane@gmail.com	Jane
alice@gmail.com	Alice

3rd Normal Form

✖ 2NF - Every column must depend on the **whole primary key**, not part of it

✓ Columns should not depend on non primary keys

- If a column depends on another column that is not the primary key, it will violate 3NF
- Columns that depend on non primary key columns should be extracted to their own table using the depended upon column as a primary key

Apply 3rd Normal Form

order_id	product_id
1	1
2	2
2	3
3	4
4	5

order_id	email
1	john@gmail.com
2	jane@gmail.com
2	jane@gmail.com
3	john@gmail.com
4	alice@gmail.com

email	customer_name
john@gmail.com	John
jane@gmail.com	Jane
alice@gmail.com	Alice

3rd Normal Form

✖ 1NF - Each row is uniquely identifiable (no duplicate rows)

✓ 2NF - Every column must depend on the **whole primary key**, not part of it

✓ Columns should not depend on non primary keys

Apply 3rd Normal Form

order_id	product_id
1	1
2	2
2	3
3	4
4	5

order_id	email
1	john@gmail.com
2	jane@gmail.com
3	john@gmail.com
4	alice@gmail.com

email	customer_name
john@gmail.com	John
jane@gmail.com	Jane
alice@gmail.com	Alice

3rd Normal Form

- ✓ 1NF - Each row is uniquely identifiable (no duplicate rows)
- ✓ 2NF - Every column must depend on the whole primary key, not part of it
- ✓ Columns should not depend on non primary keys

Unnormalized Data

customer_name	product_name	supplier	supplier_location	price	brand	order_id	email
John	Laptop	XYZ Electronics	Toronto	800	Dell	1	john@gmail.com
Jane	Smartphone	ABC Gadgets	Montreal	600	Samsung	2	jane@gmail.com
John	Camera	XYZ Electronics	Toronto	300	Canon	3	john@gmail.com
Alice	Chair	XYZ Furniture	Vancouver	100	Herman Miller	4	alice@gmail.com

---

Normalized Data

order_id (PK)	email (FK)
1	john@gmail.com
2	jane@gmail.com
3	john@gmail.com
4	alice@gmail.com

order_id (PK)	product_id (PK)
1	1
2	2
2	3
3	4
4	5

product_id	name	price	brand	supplier_id (FK)
1	Laptop	800	Dell	1
2	Smartphone	600	Apple	2
3	Smartphone	600	Samsung	2
4	Camera	300	Canon	1
5	Chair	100	Herman Miller	3

email (PK)	name
john@gmail.com	John
jane@gmail.com	Jane
alice@gmail.com	Alice

supplier_id (PK)	name	location
1	XYZ Electronics	Toronto
2	ABC Gadgets	Montreal
3	XYZ Furniture	Vancouver

Normalized Data ERD

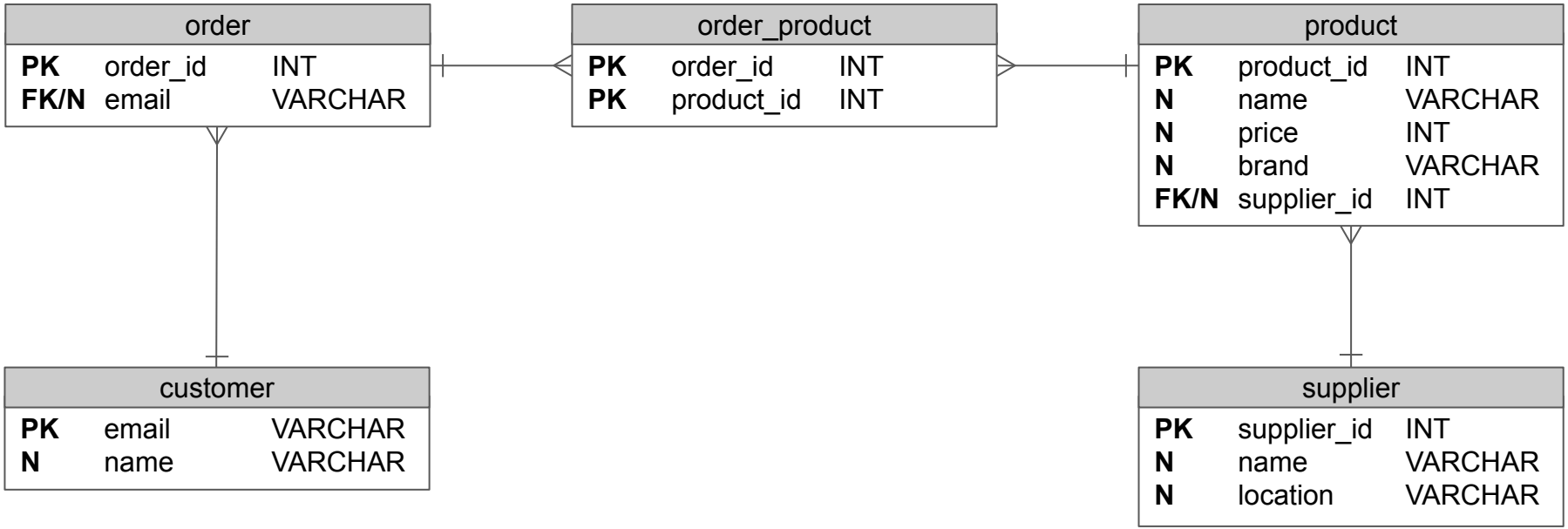


Table Relationships

- Determining the relationships between tables helps you ensure that you have the right tables and columns
- When a one-to-one or one-to-many relationship exists, the tables involved need to share a common column or columns
- When a many-to-many relationship exists, a third table is needed to represent the relationship.

Relationships will fall into 1 of 3 categories

one-to-one relationship	one-to-many relationship	many-to-many relationship

Bridging Tables

AKA Junction or Intersection Tables

- Depending on your databases need, bridging tables can store:
  - just the relationship information

    OR
  - other related information for each unique many-to-many combination
- Depending on the situation we can use different naming styles for the tables

number	course_id
n01234567	HTTP5126-A-F24
n99999999	HTTP5126-A-F24
n00000000	HTTP5126-B-F24
n01234567	HTTP5126-B-F24
n99999999	HTTP5122-A-F24
n98765432	HTTP5122-A-F24

student_course

student_number	course_id	date_enrolled
n01234567	HTTP5126-A-F24	2024-06-11
n99999999	HTTP5126-A-F24	2024-07-02
n00000000	HTTP5126-B-F24	2024-09-01
n01234567	HTTP5126-B-F24	2024-06-12
n99999999	HTTP5122-A-F24	2024-08-29
n98765432	HTTP5122-A-F24	2024-09-02

enrollment

Why use Bridging Tables

- many-to-many relationships require a **bridging table** to relate them, if not…

- Repeated data is a data redundancy
- Compromises data integrity
- Inefficient querying

number	name	course_id
n01234567	Auston Matthews	HTTP5126-A-F24
n99999999	Vladimir Guerrero Jr.	HTTP5126-A-F24
n00000000	Natalie Spooner	HTTP5126-B-F24
n01234567	Auston Matthews	HTTP5126-B-F24
n99999999	Vladimir Guerrero Jr.	HTTP5122-A-F24
n98765432	RJ Barrett	HTTP5122-A-F24

instructor	code	section	term	student
Matthew Bebis	HTTP5126	A	F24	Auston Matthews
Matthew Bebis	HTTP5126	A	F24	Vladimir Guerrero Jr.
Matthew Bebis	HTTP5126	B	F24	Natalie Spooner
Matthew Bebis	HTTP5126	B	F24	RJ Barrett

Keys - Specify Foreign Keys

- Purpose of foreign keys is to create a relationship between two tables that:
  - Maintain Consistency
    - Ensure values in one table correspond to valid rows in another table, preventing data inconsistencies
  - Can Prevent Invalid Data
    - Blocks attempts at adding data with foreign key that does not have matching primary key
  - Can Enable Cascading Actions
    - Changes in the parent table automatically apply to the related rows in the child table

Table Constraints

- Rules and conditions applied to a columns to maintain data integrity, accuracy, and consistency
- Data must meet specific criteria
- Safeguards against data anomalies, errors, and inconsistencies
- Constraints can be column level or table level

NOT NULL - Ensures that a column cannot have a NULL value

UNIQUE - Ensures that all values in a column are different

*__PRIMARY__ and **FOREIGN KEYS** are also constraints

PRIMARY KEY - Both **NOT NULL** & **UNIQUE**

Entity Relationship Diagrams (ERD)

- Used to represent a single databases architecture
- ERDs have varying layouts but will generally show this information:
  - Entities (tables)
  - Fields (columns)
  - Data types
  - Relationships
  - Constraints

Normal Forms

First Normal Form (1NF)

- Each attribute (column) has a unique name
- Domain of attributes must not change (same data type)
- Each row is uniquely identifiable (no duplicate rows)
- Each cell must have only a single value (atomicity)

Second Normal Form (2NF) – Including 1NF

- Every column is dependent on the whole primary key

Third Normal Form (3NF) – Including 1NF, 2NF

- All non-key columns are functionally dependent on the primary key