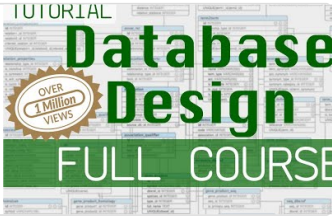
Database Design Notes

Notes on the Database Design Course by Free Code Camp

Database Design Course - Learn how to design and plan a database for beginners

This database design course will help you understand database concepts and give you a deeper grasp of database design. Database design is the organisation of data according to a database model. The designer determines what data must be stored and how the data

https://youtu.be/ztHopE5Wnpc



Author: https://github.com/shy-tan

Credits to Abuzar Qasim for first two chapters

PDF: https://github.com/shy-tan/skipq-stuff/blob/main/dbms_notes.pdf

Day 1 - Introduction

Database

- Something that is used to store data

Data

- Anything that has some sort of value

Spreadsheets vs Databases

Spreadsheet	Database
All the data is stored in a single file in tabular form (rows and column)	The data in database can be stored and organized into different tables.
Processing time of spreadsheet goes down as the data volume increases	Better performance than spreadsheets when the data volume increases
Filtering and querying spreadsheet is not easy	Filtering and querying is easy
All data or no data	Data can be divided across multiple tables
Less secure	We can allow different users to access different data

Relational Database

- A type of database that stores and provide access to data points that are related in one

another.

- Relation in relational database is basically a table consisting of attributes of real

worlds entities.

- Each row in a table represents a specific entity for that entity type (an example of entity type could be **USER** and entity could be **AHMED**. A row is called a tuple.
- Columns are the attributes

Entity

- Anything we store data about

Attributes

- Attributes are the things we store about any entity / entity type.

Database Management System DBMS

→ Relational Database Management System is a sub category of a DBMS designed to be used on relational data.

- DBMS system basically uses our data, manages it and allow us to view it in a human friendly way.

- Allows us to query, change the appearance (view mechanism) and change the way our data is stored

- **Queries** are basically the operations we can run on our data like searching, inserting, updating deleting etc.
- **View Mechanism** allows us to change the surface appearance of the data.
  - Create new views based on the original data
  - Views provide us with a layer of security Security mechanism like authorization and authentication are also provided by DBMS. For example allowing only few users to update and delete data etc.

- **RDBMS** allows us to do **Transactions**

- **Transaction** is basically some operation we perform on our data . But it is either complete all the way or it doesn’t work at all , like banking transactions

Introduction to SQL

SQL

- Programming language that is used to communicate to database
- Not specific to any RDMS
- Two sub categories of SQL
  - **DDL** Data definition language: Used for defining structure of a database, including all the connections between data/tables. Include commands like CREATE etc.
  - **DML** Data manipulation language: Used for manipulating data within a database. Based on commands like INSERT , UPDATE , DELETE etc.
- **Joins** are basically used to join multiple tables to create new views etc.

Database Design

- Database design is basically a collection of steps that help us create, implement, and

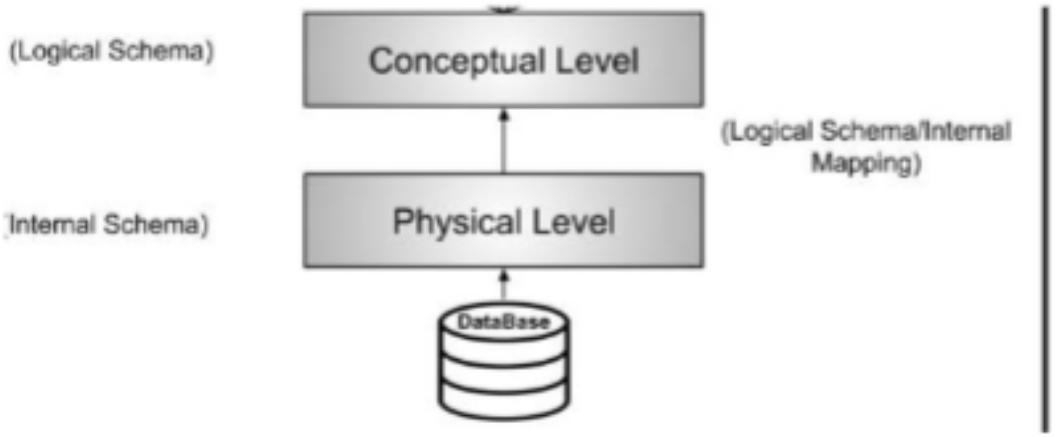
maintain a database

- A method to store separate information over multiple tables , rather than having to

store in one huge table etc.

- A design of database can be classified as good or based Data Integrity

- A database design is usually broken down into 3 steps (can vary) :
  - Conceptual Design/ Schema
  - Logical Design/ Schema
  - Physical Design/ Schema



Data Integrity

- Data integrity is basically that all your data stored in database is correct, it is up to

date , there is no disconnection to other data

Conceptual Design/ Schema

- The conceptual schema describes the Database structure of the whole database. This schema hides information about the physical storage structures and focuses on describing attributes, entities, relationships, etc.
- In conceptual schema we think about how things/entities are related etc.
- Sort of a brainstorming phase. We don’t think about limitations etc.

- Defines all database entities, their attributes, and their relationships . Like how

USER table is related to SALE table etc.

Logical Design/ Schema

- In this phase we start thinking about the tables . How the tables are structured etc.
- Structuring how our columns, data types etc.
- It specifies all the logical constraints that need to be applied to the stored data.
- The logical schema represents how the data is stored in the form of tables and how the

attributes of a table are linked together.

Physical Design/ Schema

- In physical schema we start implementing out logical design into a database.
- Specifies how the data is stored physically on a storage system or disk storage in the form of Files and Indices.
- This step involves selecting the appropriate RDBMS system .
- The type of server etc.
- Also focuses on the security constraints etc.

Day 2 - Data Integrity, Important Terms, and Relationships

Data Integrity

- The term data integrity refers to the accuracy and consistency of data.
- The correctness of data in our database
- Type of Data Integrity
  - Entity Integrity
  - Referential Integrity
  - Domain Integrity

Entity Integrity

- Entity is basically anything we store data about like USER.
- Entity integrity basically refers to unique entity. Basically entity integrity defines each row to be unique within its table. No two rows can be the same. To achieve this, a primary key can be defined. The primary key enforces that – no two rows can contain information about the same entity. Like using `user_id` / `user_name` column to make every row unique.
- Having uniqueness among rows

Referential Integrity

- Referential integrity is concerned with relationships.
- When two or more tables have a relationship, we have to ensure that the foreign key value matches the primary key value at all times. We don’t want to have a

situation where a foreign key value has no matching primary key value in the primary table.

- If two entities in different tables are connected the connection between should always exist .
- To enforce referential integrity we use foreign key constraints etc. Like if a user gets removed , remove all the comments of that user from comments table.

Domain Integrity

- Domain integrity concerns the validity of entries for a given column. Selecting the appropriate data type for columns.
- Acceptable values for the columns.
- The range of acceptable values for columns . Like phone number column should only contain 11 characters etc.
- Correct value types for correct columns etc.

Database Terms

1. Data
2. DBMS
3. Relational Database
4. RDBMS
5. **NULL : Emptiness of value in column is called a null. When a column has no value stored in it. No data in column.**
6. **Anomalies/Anomaly : Errors within our data . Anomaly is an inconsistency in the data resulting from an operation like an update, insertion, or deletion.**
7. Integrity

Design Terms

1. Entity
2. Attribute
3. Relation: Another name for table
4. Tuple : Another name for row
5. Table : Physical representation of relation.
    a. Rows : all attribute values for a entity
    b. Columns : specific attribute values
6. Record: Another name for row
7. Field : Another name for column
8. Value : Thing we store in column
9. Schema : Structure of a database
10. Normalization : Steps to get the best database design. Process of improving database design.
11. Key : Something to make everything unique among tables like `user_id`.

SQL Terms

1. SQL
    a. DDL
    b. DML
2. Front End
3. Back End
4. Client Side
5. Server Side
6. Views
    a. Joins

Atomic Values

- Values store one thing.
- Everything we store in database should be one thing. Like columns are about one thing , and the values within that columns should be one thing . storing complete name in one column is a bad design . it should be divided into two columns `first_name` and `last_name`.
- Don’t go too extreme
- Try storing smallest one individual thing.
- Break columns into what we can treat as one thing.
- Columns should be about one thing , and values inside columns should be singular. Like address should be broken down into Country , City , street address, zip code etc.

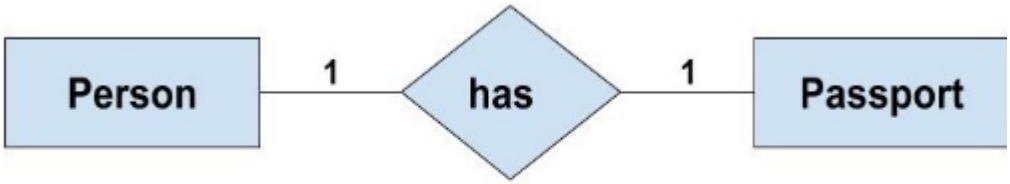
Relationships

- Relationships talk about entities and how they are related to other entities in someway.
- How different entities in a database are related to each other.
- Instead of storing all the data in one table we split it into multiple manageable tables and then create a relationship among these tables.
- There are three type of relationship
  - One to one relationship
  - One to many relationship
  - Many to many relationship

One to One Relationship

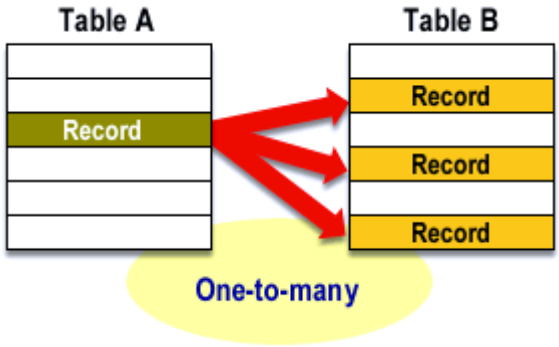
- One entity has a connection or is related to one another entity.

- A one-to-one (1:1) relationship means that each record in Table A relates to one, and only one, record in Table B, and each record in Table B relates to one, and only one, record in Table A.



One to Many Relationship

- One entity can have a relationship with many other entities but a specific entity can always be related back to only one entity . Like a user can have multiple comments but every comment will always be related to one user.



- Such a relationship exists when each record of one table can be related to one or more than one record of the other table.

Many to Many Relationship

- Multiple entities can be related to multiple entities.
- Many to many relationship are difficult to store in a relational database
- Many to many cannot be directly stored in relational database.

Day 3 - Designing Relationships

One to One relationships

1. one to one relationships are *usually* stored as **attributes**

id	name	username
6	Abdullah	dev123
7	…	…

id and username have a one-to-one relation so we put them in the same table

2. one to one relationships might also be designed as separate tables

When to store 1:1 relations in different tables?

id	fname	lname	card_no	issue_date	exp_date
…	…	…	…	…	…

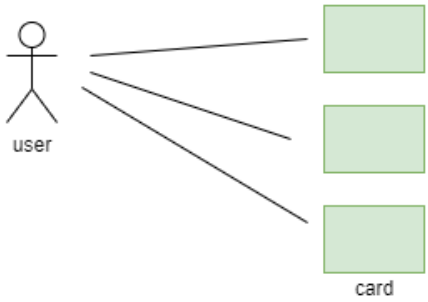
A table should be about only one thing. But the above table becomes a table that is storing information cardholders (`fname`, `lname`, `card_no`) and cards (`issue_data`, `exp_date`) so

→ divide the table so that we get different tables, each table being about only one entity (cardholders and cards in this example)

→ connect multiple tables using Foreign Keys

One to Many relationships

Lets think of an example, there are two entities: users and cards. How are they related? A user can have many cards, a card can be owned by only one user (can be owned by multiple users but lets not think of that for now)



→ user table can not hold card information because we don't know how many columns to reserve for card info.

→ card table can have multiple instances of same user (connected via user_id as Foreign Key)

card_id	user_id	…
1	**12**	….
2	244	…
3	**12**	…

(`user_id` is FK)

note: In 1 to Many relationships, Foreign Key will be placed in the “Many” table

user → reviews (FK)

user → cards (FK)

user → comments (FK)

Parent and Child Tables

→ In 1 to Many relations, a table is always a child. and another table is always a parent to that child. Keys are used to uniquely connect or relate these tables

→ Primary Key is for the Parent

→ Foreign Key is for the Child

It is important to understand who the child is and who the parent is because in future, we have to decide on placing Foreign Keys

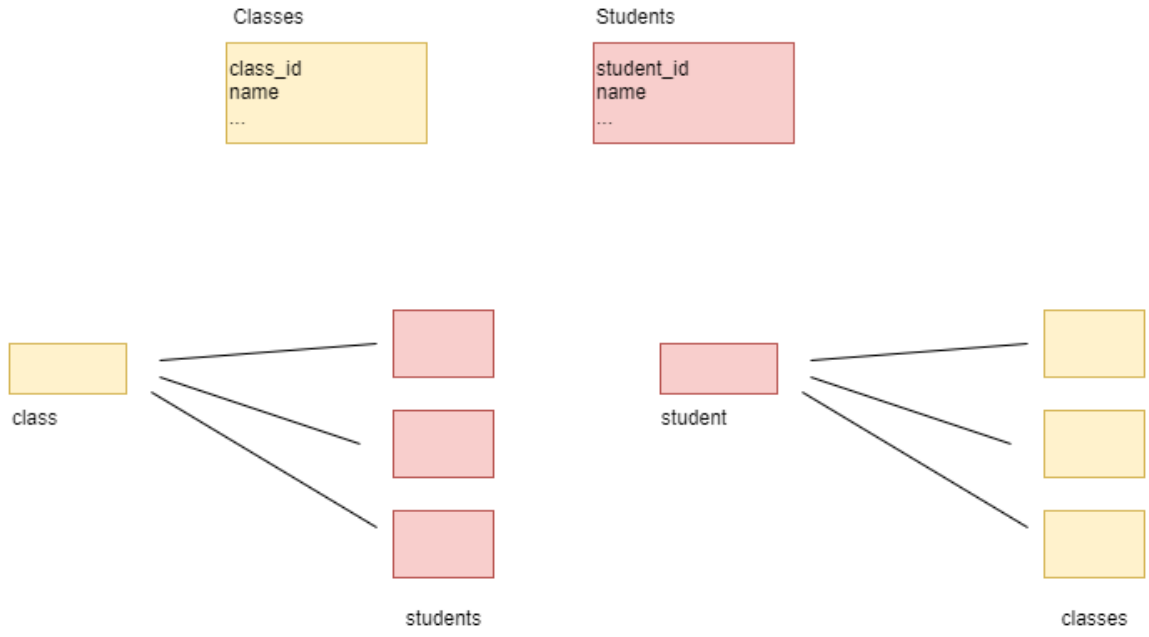
FKs - Children - Reference

PKs - Parents - Actual

Many to Many relationships

→ This happens when we cant decide who the parent is

→ Example: A student can take many classes, and a class can be taken by many students

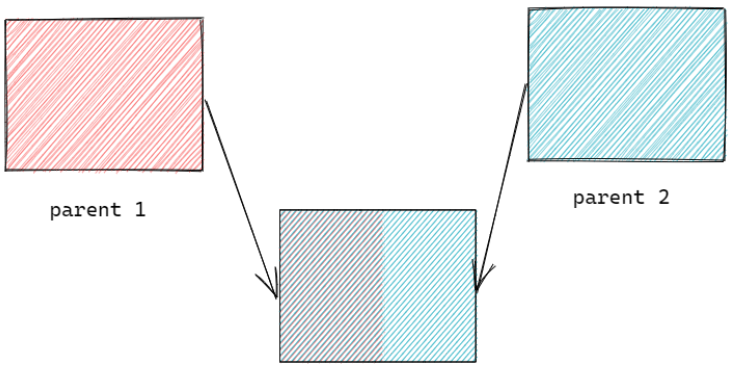


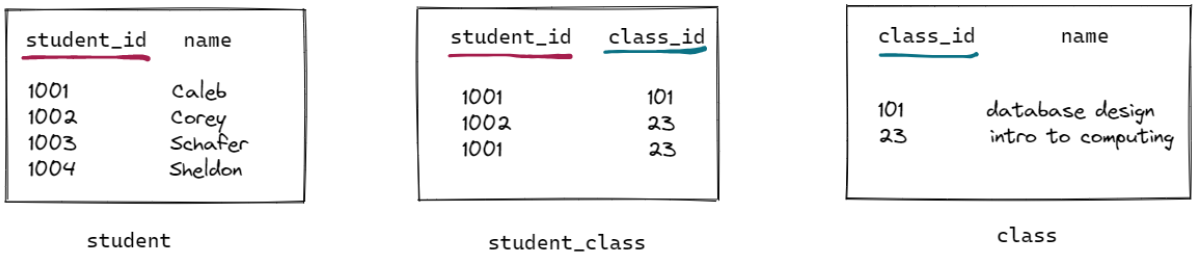
Many to Many relations are designed by splitting the many to many relation into 1 one-to-many and 1 many-to-one relation using an intermediary/ junction/ association table

→ Intermediary table contains FKs that point back to all the parents

Steps:

1. give IDs to parents (students and classes in the example)
2. use these IDs to relate both using an intermediary table





Summary

1:1	one table
1:M	two tables with child having FK pointing to parent
M:N	two parents, and an intermediary table being child to both

*

→ Keys should not repeat in a table where they are meant to be the PK, BUT, they can repeat in a table where they act as FK

For eg, `student_id` should appear only once in student table (since its the Primary Key in that table), however it can appear multiple times in the `student_class` table since its not the PK there.

→ In the `student_class` intermediary table, the Primary Key is `student_id` + `class_id` combines (called composite key)

(see how 1001 appears multiple times, because it can 🙂)

BUT

the composite key combination (`student_id` + `class_id` is always unique) ⇒ so **1001 + 23** can appear only once.

Day 4 - Keys

Keys

Good keys are

- unique
- never changing
- not null

Indexes

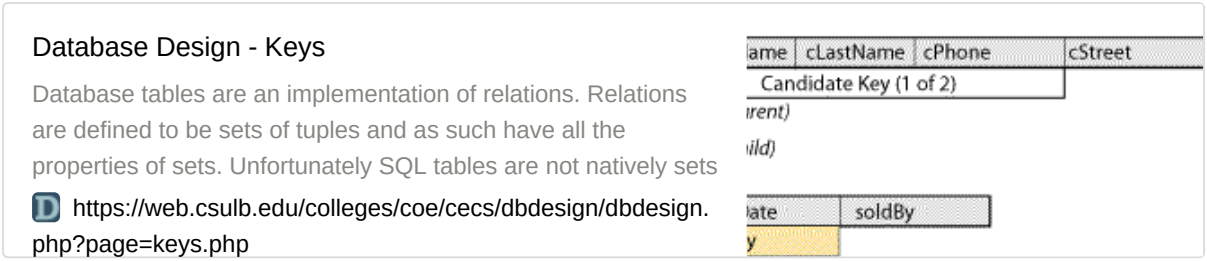
→ Keys are a kind of indexes

→ Keys are used as indexes when we use statements like SELECT x **WHERE** y

Why use keys:

1. Data Integrity ~ less maintenance ~ less incorrect data
2. Ensures uniqueness
3. Improves functionality/ sped etc.
4. Less work (while updating etc.)
5. Allows for added complexity while avoiding repeated data

https://brainly.in/question/36800276



Super and Candidate Keys

→ PK and FK has two main types

1. Super Key
2. Candidate Key

Super Key	Candidate Key
ANY no. of cols that forces each and every row to be unique	**LEAST** no. of cols that forces each and every row to be unique

- The notion of SKs and CKs is not so much in practical usage (esp. SKs) - these are to be kept in mind when designing a database for better designs

Design process

1. Ensure that: can every row be unique (SUPER KEY)

   If answer is yes, thats it! we don't need to figure out what cols ensure this thing etc.

2. Figure out how many cols are required for each row to be unique

   The minimum number is the number of your candidate keys

3. Out of the available CKs, pick the best suited key as your PK

Primary Key

should be

- **Unique**
- **Never Changing**
- **Never NULL**

Alternate Keys

⇒ All other Candidate Keys, which are not the Primary Key, are called ALTERNATE KEYS

- Alternate Keys, rarely, can be used as the index (If you get a really good AK, you might want to index it for good design measures)

Day 5 - Types of Primary Keys

→ These keys are there for us to keep in mind while designing our db, we don't need to define/ mention them

Surrogate and Natural PKs

Surrogate	Natural
not present in the table initially	present already in table
has no real world value	has some real world value or meaning

Which one to use?

→ Pick one and stick with it throughout the database

⇒ Surrogate Key SHOULD NOT have a real world meaning

Foreign Keys

→ Connect multiple tables

→ Ensures data integrity

→ References a Primary Key

→ Can reference a Primary Key that can be in the same or a different table

→ Every table has only one Primary Key* BUT a table can have multiple Foreign Keys

→ Primary Keys can never change, Foreign Keys can!

→ Foreign Keys can be or can not be NULL - depends on the use case and design:

> A table for storing information about teachers and the classes they are taking → `teacher_id` **CAN BE NULL** if the teacher is not available or left the job etc.

> A table holding information about credit cards **SHOULD NOT BE NULL** for any `card_owner_id` if we assume that we only have information about those cards that have an active owner.

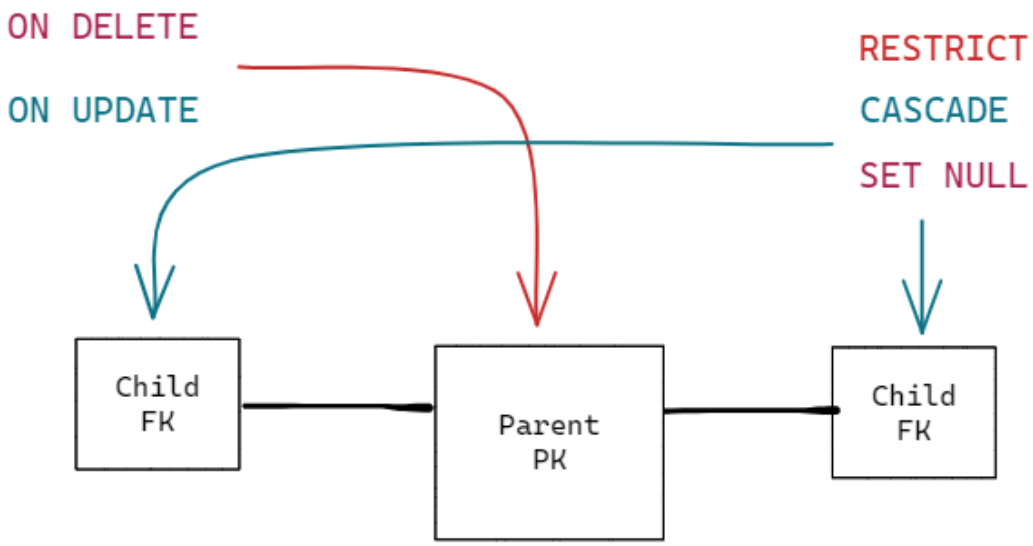
*****

doesn't matter if its a composite key ~ the PK is still technically only one key that spans multiple cols/ attributes)

Foreign Key Constraints

Required to maintain the integrity (Referential Integrity) of a database.

idea: when we **delete** or **update** parent, we want child to do something in response



RESTRICT – Throw error
CASCADE – Delete (ON DELETE) / Update (ON UPDATE)
SET NULL – Set children to NULL

RESTRICT	No action in child table/ throws error
CASCADE	Do what the parent did
SET NULL*	Set child to NULL

*

For SET NULL to work, children should not have the NOT NULL constraint

Simple Primary Key

→ Keys consist of one column for example `username`

→ Surrogate keys are mostly simple keys

Composite Primary Key

→ Keys consist of two or more columns for example `first_name + last_name + email`

→ Natural keys are mostly composite keys

Compound Primary Key

→ consists of two or more keys that uniquely identify an entity

→ similar to composite key

In SQL, is it composite or compound keys?

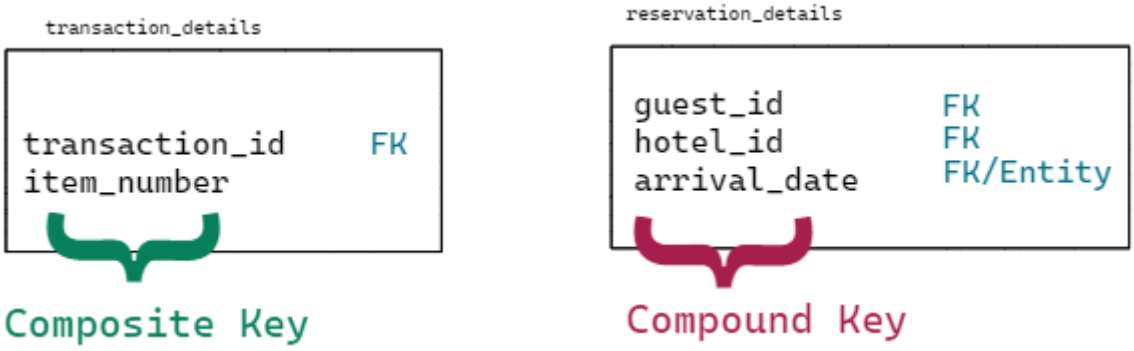
I'm still not sure why http://en.wikipedia.org/wiki/Compound_key was not consulted. It very clearly states (and is correct): In database design, a compound key is a key that consists of 2 or

https://dba.stackexchange.com/questions/3134/in-sql-is-it-composite-or-compound-keys

In SQL, is it composite or compound keys?

I'm still not sure why http://en.wikipedia.org/wiki/Compound_key was not consulted. It very clearly states (and is correct): In database design, a compound key is a key that consists of 2 or

https://dba.stackexchange.com/a/3136



⇒ keys in a composite key can or can not be FKs

⇒ all keys in a compound keys are FKs or Entities

in other words

⇒ in a composite key, at least one column is not a key

⇒ in a compound key, all columns have to be keys

Assume a table that stores information about user comments on videos in an online site. The composite key can be `user_id + video_id + timestamp` → timestamp is not a key but it uniquely identifies a user comment along with the other two keys.

Day 6 - ER Modelling

→ ER Modelling is a standard for drawing databases

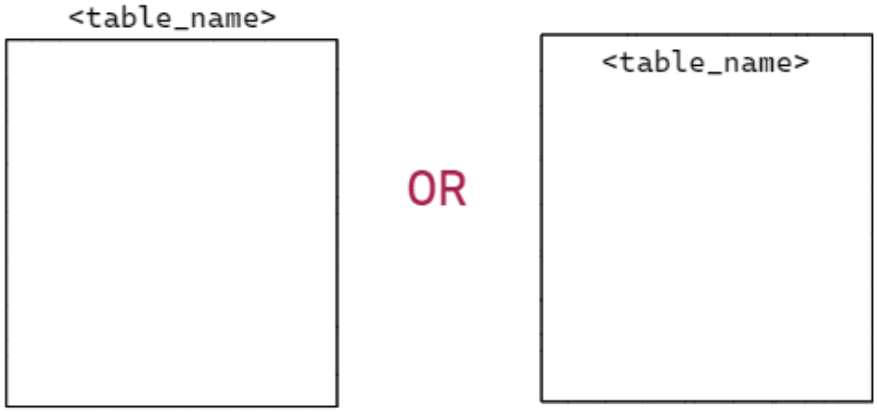
→ aka EER, ERD, ER Model, db design etc.

→ defines database structure ~ DDL

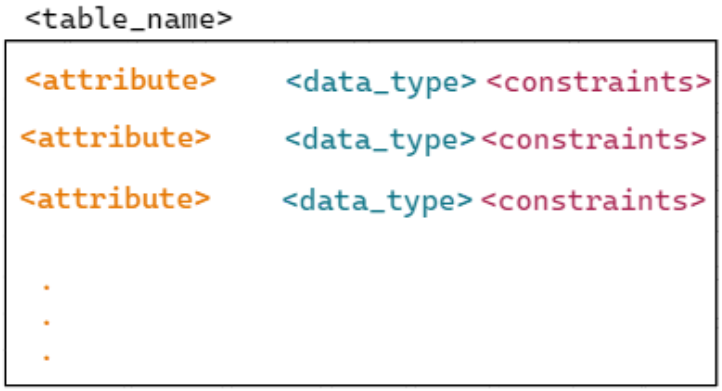
1. Tables are shown by drawing squares/ boxes
2. Table names can be placed above the table or at the top inside the table **1**
3. Attribute names along with their data types and constraints are mentioned inside the tables going from top to bottom **2**

(attribute names are essential, the other two are optional but mentioning them is a good practice)

1



2



4. Tables are connected with lines to illustrate relationships among the tables
5. Indexes and Foreign Keys can also be mentioned in a table if you want to

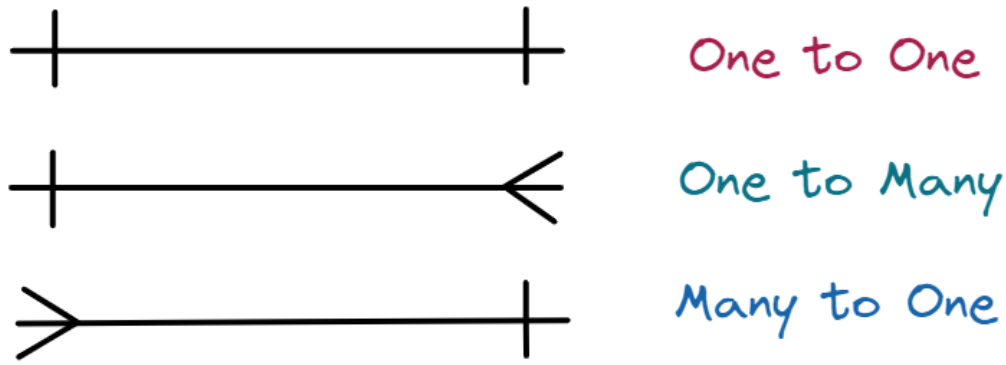
Cardinality

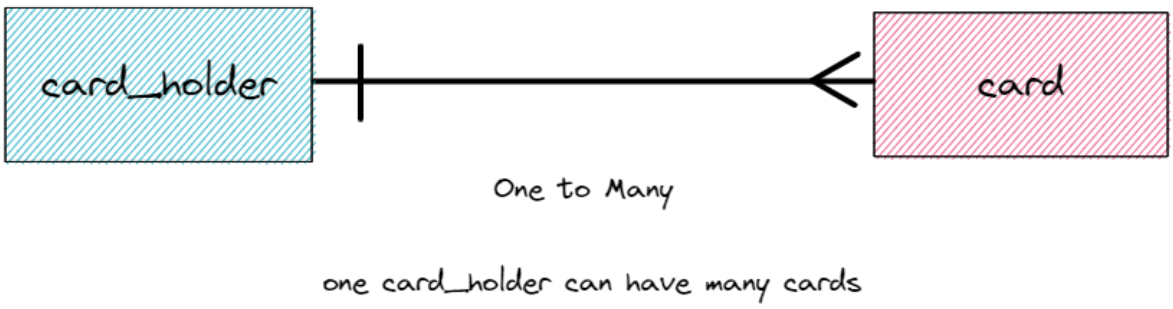
→ Cardinality refers the type of relationship between tables

→ Cardinality refers the type of relationship between tables

→ These can be One to One, One to Many, and Many to One

→ These relations are shown by drawing lines as follows





one card_holder can have many cards

Modality

→ Modality describes whether or not a child is a required child

→ If the NOT NULL constraint is present, the child would be required to have a record in every single row, and vice versa

Normalization

→ Normalization fixes data integrity, redundancy issues

→ Normalizing a database can be thought of as a checklist that you follow while designing your database, and when you get to the end of of that list, you have a good design

> Normalization is a systematic way that we follow to produce a good structured database

→ 3 types:

1. First Normal Form 1NF
2. Second Normal Form 2NF
3. Third Normal Form 3NF

Note: Normalization is a systematic process, you can't jump to 2NF if you're not currently in the First Normal Form

1NF

- **atomicity**: data should be atomic

⇒ imagine a column `address` having values like **16, Murree Road Near Defense Sectt.#2, Rawalpindi**

This is problematic:

1. This address can be decomposed into city, sector, street etc.
2. If you want to run queries such that you want to filter users that are located in sector x city y, you'd have an overhead of parsing the string properly before querying data

Solution:

→ Store individual/ atomic parts of the complete address in separate columns

⇒ Now imagine a table having multiple values for a column

id	user	email
101	dev123	a@b.com
101	dev123	b@c.com
102	...	...

Note that the values inside shaded region are repeated data because a user wants to have multiple emails

BUT

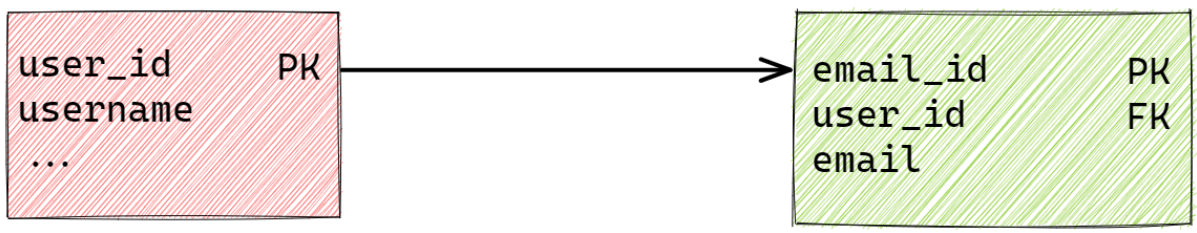
we know Primary Keys (id in this example) SHOULD NEVER REPEAT

why to avoid repetition?

If down the road we want to update some attribute for this user, we'd have to update every record of this user (and there's a risk of us missing/ forgetting to update some records) → **Data Integrity issues**. :x

Solution:

→ Make another table, which only stores emails and use the **id** of user as FK in this table, now the user can have as many emails as he/ she wants.



Note: PKs should never repeat, FKs can repeat

2NF

→ Deals with **partial dependency**

→ **Partial Dependency**: happens when a column depends only on a part of the Primary Key

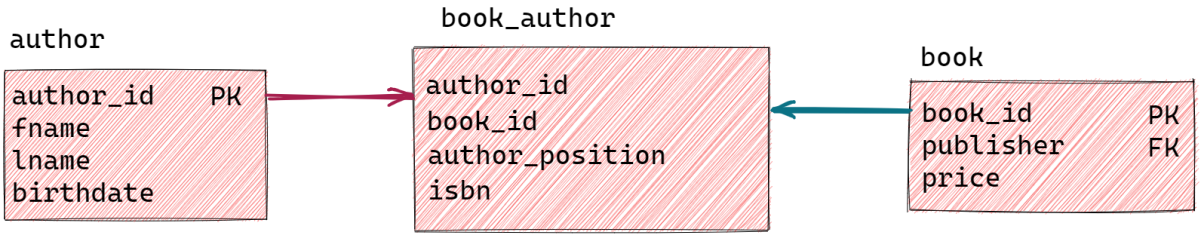
… this is common with many to many relations having intermediary tables and Surrogate Keys

→ Primary Key consists of two or more columns in this case (Compound/ Composite Primary Key)

…. *because if you only have 1 column as Primary Key, then you can not depend on half of it ;)*

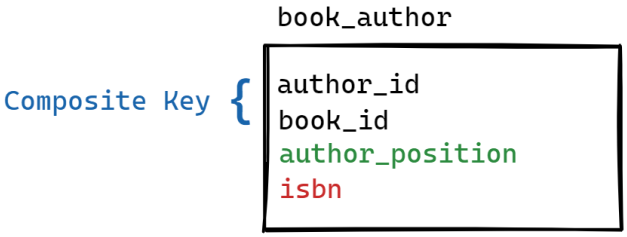
⇒ Imagine a table Person, having columns/ attributes `person_id, phone, name` with `person_id` as the PK. In this table, `phone` and `name` both are dependent on `person_id` in order to uniquely identify a person.

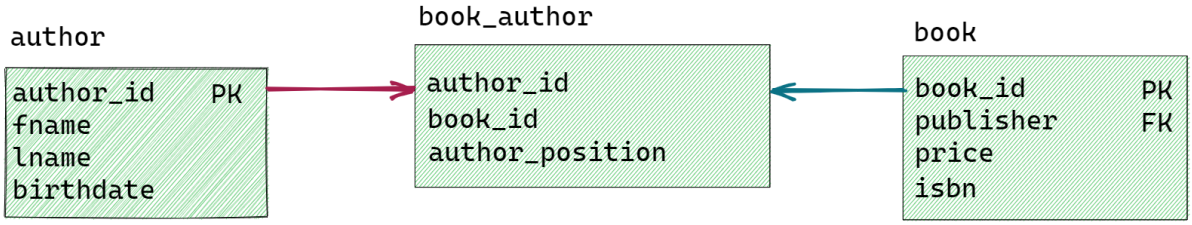
⇒ Imagine the below many to many relationship design



BEFORE

Second Normalized Form





AFTER

Second Normalized Form

In the above example, `book_author` intermediary table has these attributes/ columns: `author_id, book_id, author_position, isbn`. Amongst these, `author_id` and `book_id` are the PK (Composite Key), `author_position` is an attribute that describes the rank or position an author has on the book cover when there are multiple authors. So, `author_position` does depend on the Author entity, however if we think about it, an Author does not have the same position on all his books. So, `author_position` is better suited to be placed in the intermediary table than the author table since this attribute depends both on the author and the book, in other words it has a dependency on both `author_id` and `book_id`.

Similarly, if we think about `isbn`, the ISBN number of a book always remains the same, it has no relation with the book author. `isbn` has dependency on `book_id` but not on `author_id`. So, `isbn` should be placed in the book table.

Solution:

→ Take Partial Dependency, move it to a suitable table and reference it with a Foreign Key.

For a database to be in 2NF:

1. It should already be in 1NF
2. It should not have any partial dependencies

Note: If you are in a table where the PK consists of only one column, then you are already in the second normal form!

3NF

→ Deals with **Transitive Dependencies**

→ **Transitive Dependency**: A column depends on another column which depends on the Primary Key

why is transitive dependency bad?

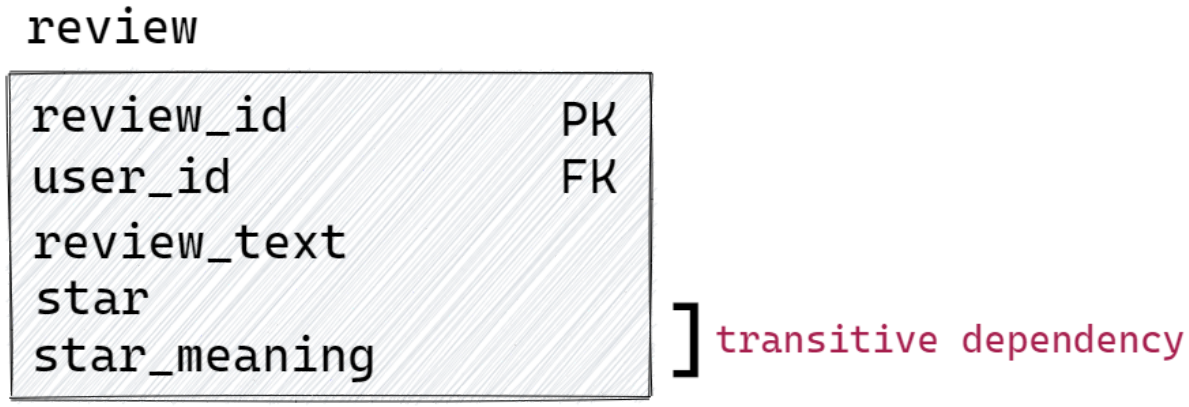
→ leads to redundant data

What is wrong with a transitive dependency?

One way to express the 3NF is: All attributes should depend on the key, whole key and nothing but the key. The transitive dependency X->Y->Z violates that principle, leading to data

https://stackoverflow.com/q/9950367/10934636

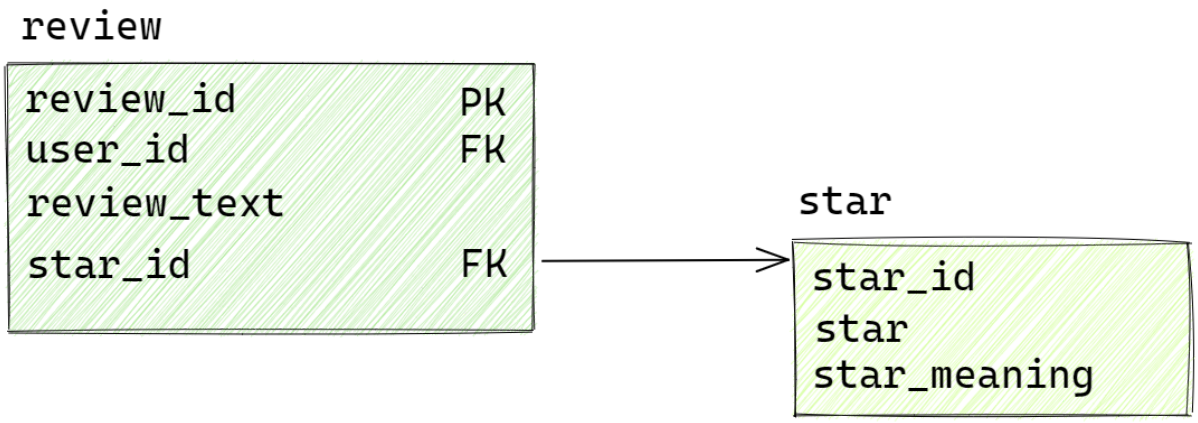
⇒ Consider the below table



`star_meaning` depends on `star`. In other words, if the value of `star` changes, the value of `star_meaning` also changes. Also, `star` depends on the `review_id` to uniquely identify a review.

Solution:

→ Put the columns that are creating the issue in a separate table, and refer them using an FK in the original table



For a database to be in 3NF:

1. It should already be in 1NF
2. It should already be in 2NF
3. There should be no transitive dependencies

Summary

1NF	make everything atomic
2NF	remove partial dependencies
3NF	remove transitive dependencies

Day 7 - Indexes, Data Types and Joins

Indexes

Clustered Index

A type of index that reorganizes data. For example, a contacts list reorganizes contact numbers alphabetically

Non-clustered Index

A type of index that simply points to the data without any kind of reorganization. For example, the index or table of contents in a book only mention the page numbers for faster access but dont reorganize data

⇒ We can have multiple non-clustered indexes

⇒ We can have only 1 clustered index since it actually reorganizes data that way

When/ what to index?

A column that you use many times on a regular basis while querying data and you want these queries to be super fast ⇒ index it!

However, there’s a downside to indexing, whenever the data updates, index should also be updated

Composite Index

An index that spans across multiple columns

Data Types

Every column has to have one of the available data types that your DBMS allows → so data types slightly differ based on what database you use!

Generally, these data types are divided in three categories

1. Numeric
2. String
3. Date

integer, float, timestamp, varchar are some sub-categories belonging to the above three categories

Joins

Joins are way to present data that spans over multiple tables, in a user friendly way

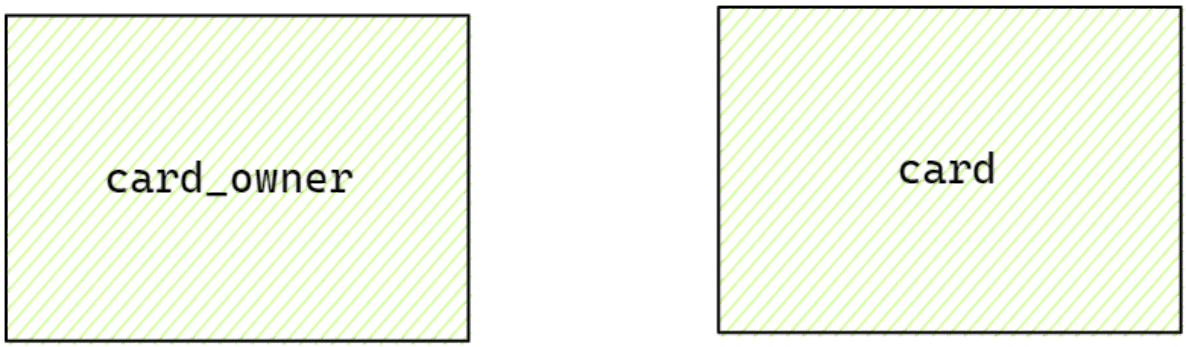
→ Joins are related with DML operations

⇒ Joins do not change the structure of a database, the just change they way data is presented

Types of Joins

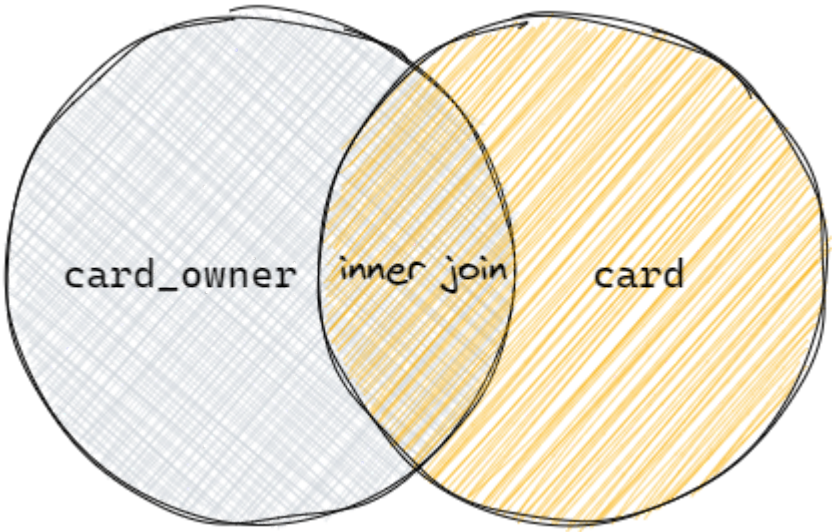
Inner Join

⇒ Joins two (*or more*) tables and get only that data that exists in all the tables that are joined



If we perform an inner join on the above two tables, it can be expressed as follow

1. Get all card owners that have a card
2. Get all cards that have an owner



⇒ The inner join operation will get us the common data found in both tables. It can be thought of as an intersection in two sets, as shown in above Venn diagram

Sample code

```
SELECT fname, lname, exp_date
FROM card_owner
INNER JOIN card
ON card_owner.owner_id = card.owner_id
```

Inner Join on 3 tables

→ Same idea as inner join for two tables

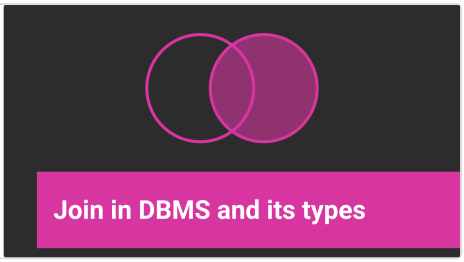
→ An inner join on 3 tables will first perform an inner join on the first two tables

→ The resultant table then undergoes an inner join operation with the third table

What is Join in DBMS and what are its types?

In practical scenarios, whenever we make use of DBMS, we deal with multiple database tables. Actually, in most cases, we need to combinedly work on these tables. We have to use the

https://afteracademy.com/blog/what-is-join-in-dbms-and-what-are-its-types



SQL JOIN Cheat Sheet

Download this 2-page SQL JOIN Cheat Sheet in PDF or PNG format, print it out, and stick to your desk. The SQL JOIN Cheat Sheet provides you with the syntax of different JOINs, and

- https://learnsql.com/blog/sql-join-cheat-sheet/