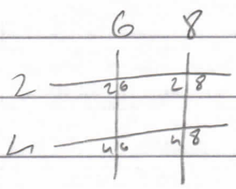
UTUBE

Database Design

Caleb Curry



(ent) Entity — הלקוח שלו אותו רוצים לשמור נתונים

(att) Attribute — ex: name, username, password

type: user — entity

name	username	password
Joe	joebayd3	pie45#!
Sheli	sheli345	gem34!3

Row: ה- att של ent ספציפי — record, tuple

Coulom, field, Att: ה- value של att ספציפי

DBMS: Data Base Managment System — 00:30

SQL — Structure Query Language

שפה שעובדה: לתקשר עם database

(DDL) Define —then→ Manipulate (DML)

structure — insert, delete, Sort, Search, etc...

Database Design

כללי

Brain Storming כלל ← Conceptual — לא חלק של הטבלאות, על לוח הרצון

Dictionary בנית ← Logical

באיזה שרת נשתמש, איך אנשים יקבלו גישה? ... ← Physical — התקנת השרת, מי תהיה גישה, test, Security

ספציפי

Data Integrity

! בעיה: אם יש ערכים מאותו Att כה יותר מפעם בטבלה, לדוגמה address, address. אתה מעדכן אחת ואת
לא את המקבילה לכן יהיו 2 כתובות. מי היא הנכונה?

types: Entity

Ref: Between 2 tables or 2 entites

Domain: range of values: Don't accept values that a out of range

char(20), phoneNo: accept only numbers!

Defenitions

data: what to keep track of — database: where to store

Relation DB: everything is in tables

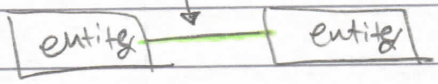
RDBMS — DBMS: Managment of

Null: There is no data in this field

caleb 1

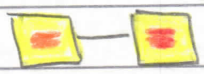
Anomalies: thing that are unexpected
Integrity: implemented to prvent
Domain: range of values that are exacptable to store within a column
synonem: File = table Record = row Field = Colum
Normalization: step to follow to have better DB design
Atomic value: 1 thing 1 att = 1 value name = פיני זרעיה (2 values) => non atomic

Relationship - Designing DB



one to one # one to many # many to many

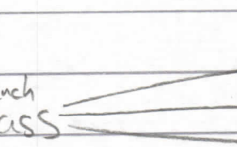
one to one



husband ↔ wife: 1 husband could have only 1 woman and vice verca

one to many

many to many



Desigining Relationships

- **1:1**

מקרה 1: משתמש ← שם משתמש

{ ID | Name / 531 | Oren } 1:1 username {oren345H}

ה-username צריך להיות att של היוזר, ולא כ-relationships איתו

ID	name	username
531	Oren	oren345H

מקרה 2: משתמש ← כרטיס אשראי. נניח שלמשתמש יכול להיות רק כרטיס אשראי אחד ולא יותר

Bad

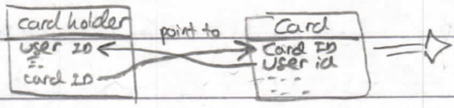
cardholder
- ID
- name
- lastname
- card.no
- card.date
- card.exp

משתמש ← כרטיס אשראי ← תאריך הוצאה, תאריך תפוגה, מסגרת

כל הנתונים בירוק מצביעים על ה-holder. הנתונים באדום מצביעים על כרטיס האשראי. זה מצריך לשים אותו בטבלה משלו.

כלל: אם אני עושה תכונות פנימיות, בניגוד ל-att ויוצא עם תכונות ביחסים של 1:1 זה אומר שה-att צריכה טבלה משלה.

good Design



⇒ מספר שלה לא משהו } עדיף שרק ה-card יצביע ל-Holdercard ולא הפוך

מה הבעיה ב-Bad, למה לא לשים כרטיס של ה-card וכרטיס של היוזר באותה הטבלה?

- תשובה: במקרה כזה (3) רק לקרוא לטבלה user&card. וזה הפרה של rule of 1: 1 entity = 1 table

- Bottom line

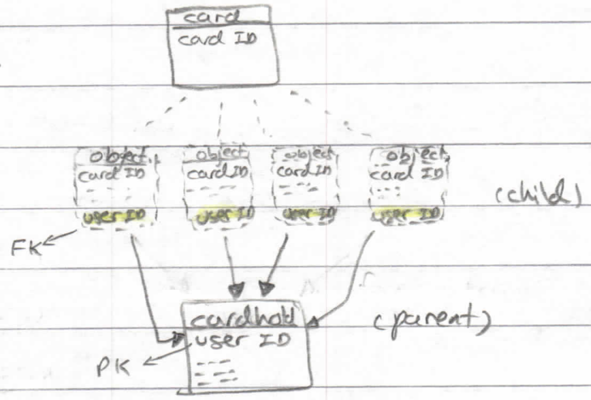
most of the time 1:1 relationtip will result in assaigment of attribue if we need to store more data on the other side of the relationship
→ it will resulat with being a new entity (table) and not an att.

1:many → 1:N

ex: 1 card holder can have many cards

למה לא לשים את ה- cardID ב- user?

⇐ זה יוצר 1:1 relationship



PK and FK

PK: every tulpe should have a key that is called a primary key

FK: foriegn key. if you have it, you are a child and you point to the parent

Foreign key: key that connects between 2 class. it doesn't represent the class that it's in, it represent the target class. it point to the primary key in the primary table. {given to the (many) side}

children \ Parent

הילדים יורשים values מההורה אבל ההורה לא יורש כלום מהילד כיוון שהוא לא מכיר אותו.

למה לא? כי ל- parent אין FK שמצביע על הילדים. יש לו רק PK

למה זה חשוב מי ההורה ומי הילד? כי זה יעזור לנו להחליט איפה לשים את ה- FK

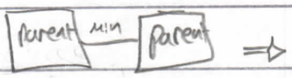
child לא יכול להתקיים בלי ה- parent שלו. user ! order

ל- order אין קיום בלי היוזר שהזמין אותה: ~~order~~ ← user = ✓ order → ~~user~~ = X

many : many   M:N

ex: class has many students and student has many classes

אי אפשר מעשית לעשות many to many ; זה אומר ש:



⇒ אין כזה דבר

עוד דוגמה: אם נניח אני רוצה לחבר class או Student

Class
- ClassID
- proName
- student1ID → אסור בהחלט
- student2ID
- ⋮

↓ מה אם יש לי 30 תלמידים?

צריך לבנות אותה מחדש → מה אם אחד מתווסף?

Student
- ID
- name
- last
- class1
- class2
- ...

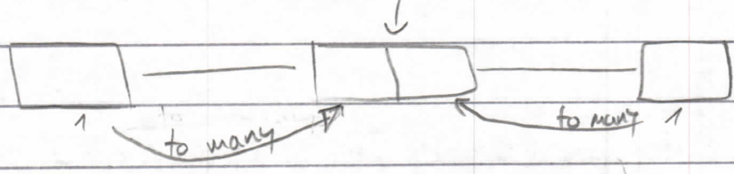
מה אם לתלמיד יש 20 class?

מה קורה אם לתלמיד אחר יש 1 רק?

↓ זה אומר ש- 19 columns יהיו ריקים (null)

⇩ BAD Design

פתרון: Intermediate Table פיצול ל-(1:N) ו-(1:N)



taking the previous example

classes: math, science, English   PK: 63, 75, 89

Student: Josh, Claine, Sally, Joni   PK: 8, 17, 16

Parent — class

class
63 ....
75 ....
89 ....

student_class (Child)

classid	studentID
63	8
75	8
89	8
75	17
63	17
89	16

Parent — Student

Student
8 ....
17 ....
16 ....

class 75, יש לו (2)? תלמיד 8 ו- 17

תלמיד 8 לאיזה הולך class? 63, 75, 89

ה- Intermediate table הוא child שמקשר בין 2 ההורים.

Keys

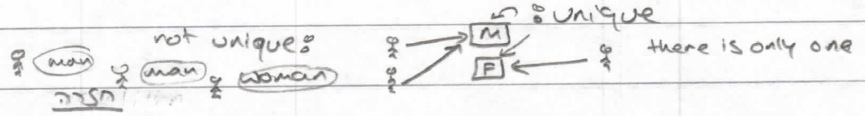
Primary key: everything unique, it should never change and never null

natual key: it's an att. that it's really unique like: username, ID number
avoid: names, lastnames, passwords, emails.

Surrogate key: made up keys, colum that is added to your table to id it.

Keys: What they do

- Integrity: protect the integrity
- Keep everything unique
- Improve functionality: less work, easy update
- Allows more complexity



superkey: more than 1 colums (attributes) that create a unique row
candidate key: the minimum number of colums needed to create a unique row
ex: username | email | name | age | adress | family name

superkey: email + username + age → that combination is a super key. every row will be unique and will represent only one user

Candidate key: username. this is the minimum att. that I need to make every row unique cause it describe 1 user and 1 user only!

1. can every row be unique? → yes ✓ I've got my superkey
2. how many colums are needed? → candidate key
3. How many candidate keys do I have? → email → username → ...
they are call candidate key because I can have a 1 or more... candidates who will candidate to be my chosen [PK]

Primary key: the key that I've chosen to make every row unique, and to ∞∞ tabels

Alternate key: All the candidates key that were not selected (as the [PK])
it can be a good practice to index them. it's an option

Look up table

if some att has limited number of values.

ex: credit card holder

one of the att. is credit_type — platinum / gold / silver / premium

Look up table: אני יכול לשים את כל אחד מה- value הללו ב- table אחר ולא צריך יותר לכתוב

לא יתקין עבודה: אם יש אחד החברה רוצה להחליף את שם הכרטיס gold ל- basic,

אני צריך לשנות זאת רק פעם אחת ← כאן

בטבלת ה-cardholder זה רק ref, הכל לא תקין. רק ה-connection לא הושפע.

card_holder (type → foreign key)

ID	name	family	adress	type
				1
				1
				3
				2
				2
				4

type

1	silver
2	gold
3	premium
4	platinum

(+) Advantages

1) protect integrity: a value that is not defined in the "lookup" → not valid

2) less maintance: if a change is needed it applied only once and not hundred of time

Natural Key vs Surragate

Natural Keys

(+)
- less data (don't need to define new data)
- natural, intuative, real world meaning

(−)
- hard to find a really good one
- artifical, doesn't have a real meaning
- ! if the value change, it's chang the key!!

ברגע שהחלטתי שאני בוחר אחד על השני, אני צריך להשאר רק עם כל ההחלטות ב- DB

Foreign key: ref to a primary key of another table. this is what keeps tene connected.

a foreign key can accept no value (null) if the relationship doesn't exist for this row

a foreign key value may change if the situation has change

user (FK)

ID	name	car
1		20
2		21
3		null
4		23

car

20	Fiat
21	מיצובישי
22	קדילק
23	סובארו

null: יכול להיות שלמישהו אין מכונית ← FK=null

update: אם למישהו היה סובארו והוא מכר וקנה מיצובישי

FK 23 → update → FK 21

FK constraints

DELETE and UPDATE (parent -ל [illegible])

we wee delete the parent or update it → we want it to impact also its children that are connected to it by a [FK]

FK Constraints

regarding parent → ON DELETE, ON UPDATE

regarding child → RESTRICT, CASCAD, SET NULL

How does that work?

if we want to delete/update the parent. what will be the child reaction?

- RESTRICT: an erreur will occur, and the action will be aborted (won't be deleted...)
- CASCADE: whatever had been done to the parent will be happen to the child
- SET NULL: it will change (at the child side) the FK to null → no ref → no relationship

the FK in question can NOT have to NULL. (it makes sense) if I do have it defined → it will not update/delete the parent. it will abort the action

natural keys — simple: one colum / composite / compound

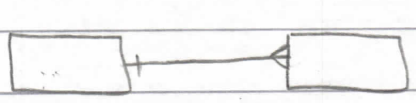
Design DB

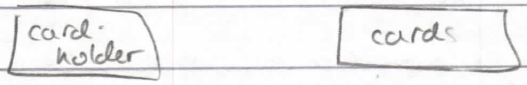
we write only the name of the colums

Cardinaly: the relationship between 2 tables

one side ——+ many side ——<

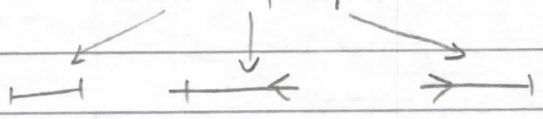
ex:





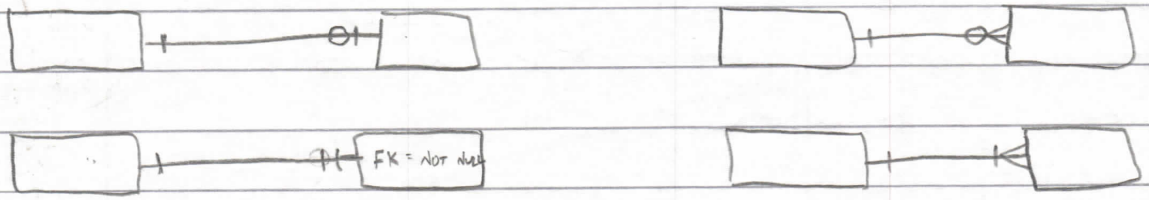
card holder 1 ... 3 cards

the only options



many cards ... cardholder 1

the next step is to decide wheater or not the FK → NOT NULL



01 : can have zero or 1 => NULL is accepted

11 : NOT NULL

Normolatizon 1NF → 2NF → 3NF

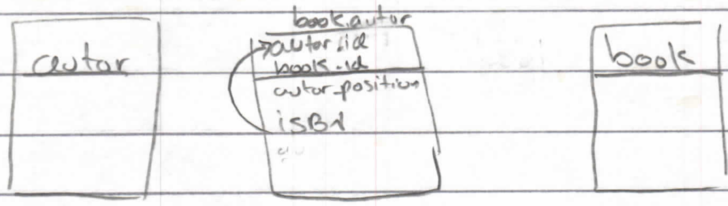
process: we go through our DB Plan and we correct things that may cause: integrity problems, reapting data...
AFTER applying the 1NF I can applay the 2NF etc..
it's a list of steps: I can't do the 2nd before I do the 1st

1NF: Data being atomic

1 colum = 1 value
1PK = 1 entity
1PK per table, no repetiton => unique
coulum should be the smallest יחידה. adress = bad → street no / city / street name

2NF: Partial depedency

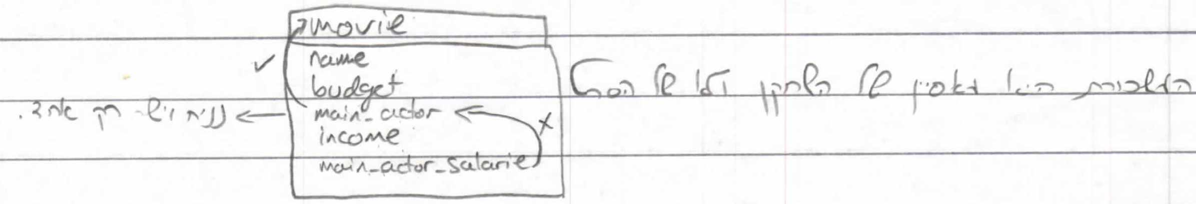
only when the PK is a composite/compound key:
it's when a coulum relates/depends/describe only one part of the key
ex: autor ⇄ book (many / many)



the ISBN depends on the book but not on the autor
solution: move it to the book table

3NF: Transitive dependancy

it's when an att depends on another att instead of the entity
ex: a movie



הדרך לבדוק היא: 2 att (2 coulums וכו') שתלויות בעמודה של transitive dependancy
אני צריך לשאול: אם אני משנה אחד, האם גם השני ישתנה? ← כן: בעיה, מפרים 3NF
← לא: הכל בסדר
הפתרון: ה- att שיוצר transitive dep ומעביר לטבלה משלו ← יקבל טבלה חדשה FK of

caleb S

Index

[+] make the query hundred time more faster, saves time

במקום לעבור על כל הנתונים כדי למצוא את מה שאני רוצה האינדקס עוזר ל-DB לדעת איפה נמצא מה.

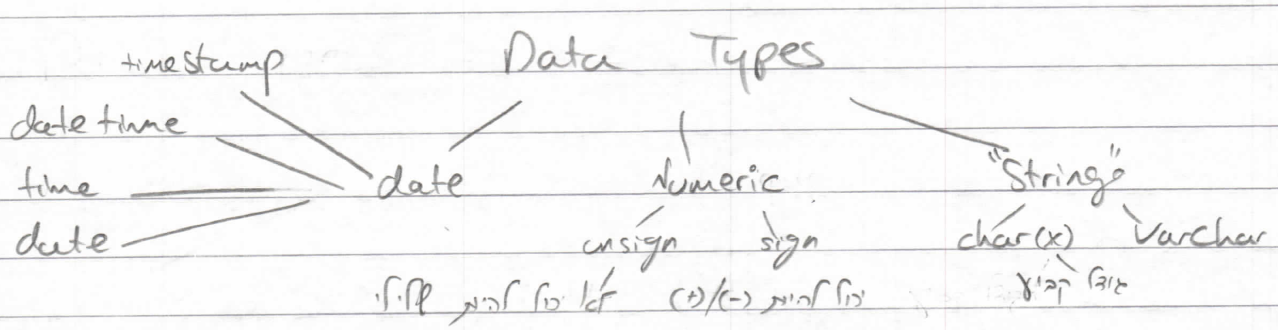
[-] everytime I update the data - I should update the index also → slow

☆ I don't want to use WHERE on an un-index coulom → can take forever

cluster index

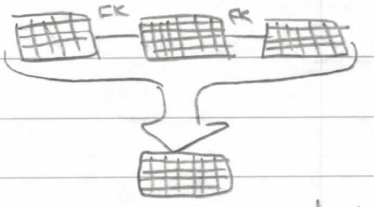
-

-



joints / 6:35 + index

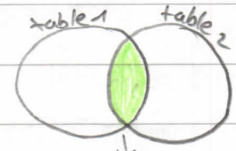
JOINS



DML - Data Manipulation Language

changing the presentation of the DB not the structure

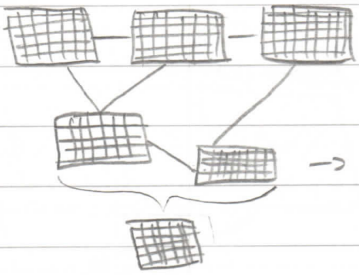
Inner Join

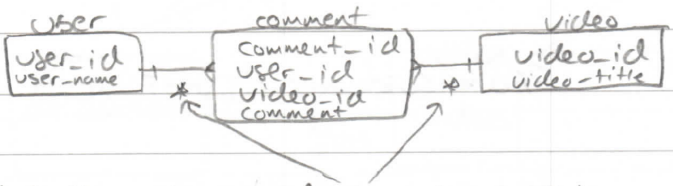


return as Inner Join

SELECT + the names of the columns we want to present
FROM table 1
INNER JOIN table 2
ON + what connect them (table1.pk = table2.fk)

3 tables





YOUTUBE

1) list the join conditions: → the connections
* user.user_id = comment.user_id
* video.video_id = comment.video_id

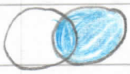
user_name	comment	video_name

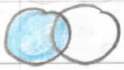
end result

```
SELECT username, title, comment
FROM user u
INNER JOIN comment c
ON u.user_id = c.user_id
INNER JOIN video v
ON v.video_id = c.video_id
```

Outer Join

- Right join
- Left join
- Full join





Right Join

returns all records from the right table (table2), and the matched records from the left table (table1). when there is no match → NULL

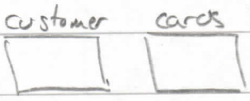
```
SELECT column(s) name(s)
FROM table1
RIGHT JOIN table2 ON....
```

What is left and what is right?

```
SELECT...
FROM ____ ⌐¬ ----> always the left table
```

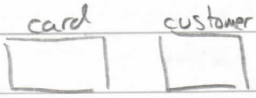
use only LEFT JOIN. why?

I want to return all of my cards



Right outer join

FROM customer RIGHT JOIN cards



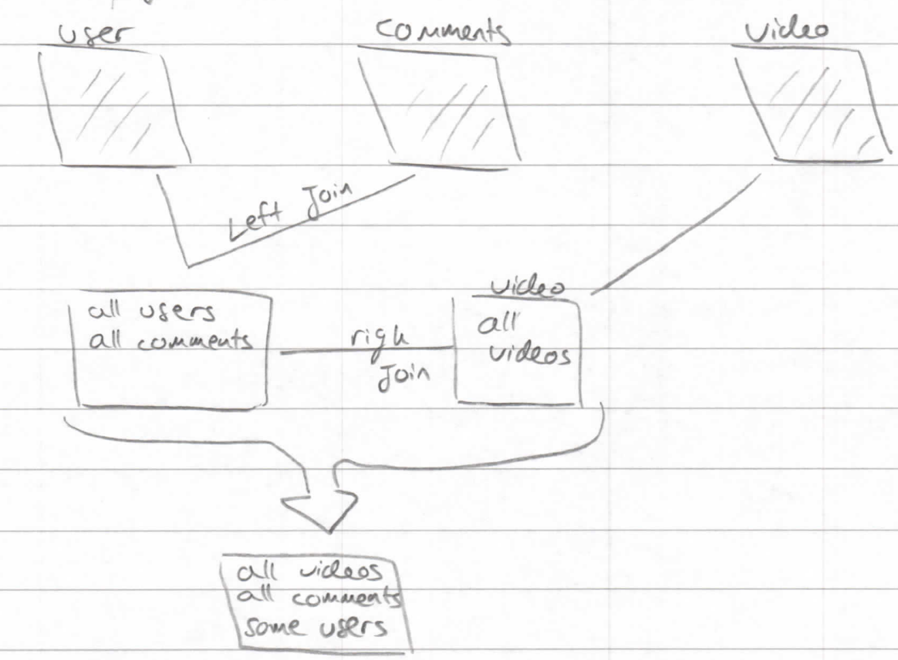
Left outer join

FROM card LEFT Join customer

Left Join

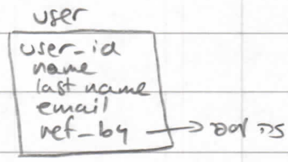
replace the word RIGHT JOIN with LEFT JOIN

Example



Self Join

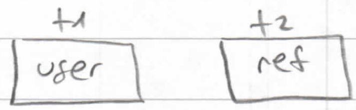
the table is joined with itself



user_id	name	last	email	ref_by
1	pini	tzru	pini@gmail	3
3	yoni	gulim	yg@gmail	null
5	dori	dor	dr@gmail	1

אני רוצה במקום להציג את הת"ז ref_by, להציג את האימייל של אותו בנאדם:

name	last	email	Ref email
pini	tzru	pini@gmail	yg@gmail.com
~~yoni~~	~~gulim~~	~~yg@gmail~~	~~null~~
dori	dor	dr@gmail	pini@gmail



אני צריך לחשוב על הטבלה כ-2 טבלאות

```
SELECT t1.name, t1.last, t1.email, t2.email AS "Ref email"
FROM user t1
JOIN user t2
ON t1.ref_by = t2.user_id
```

we must use alias!

caleb7