DATABASE CONSTRAINTS

INTEGRITY CONSTRAINTS

→ are a set of rules that ensure that data insertion/updating/other processes have to be performed in such a way that data integrity is not affected.

→ determine what tuples can be stored in the database

→ Instances that satisfy the constraints are called legal

→ We will look into the special cases of more general constraints:

1) FUNCTIONAL DEPENDENCIES (FD)

The values for X attributes determine the values for Y attributes

→ are constraints of the form $X \rightarrow Y$ where X, Y are sets of attributes

→ SET SEMANTIC: A relation R satisfies $X \rightarrow Y$ if for every 2 tuples $t_1, t_2 \in R$

$$\pi_X(t_1) = \pi_X(t_2) \Rightarrow \pi_Y(t_1) = \pi_Y(t_2)$$

If the 2 tuples agree on the value that they assign to attributes in X...

...then they must also agree on the value they assign to attributes in Y

→ FDs are trivial $(X \rightarrow Y)$ when $Y \subseteq X$

→ i.e.

R

Employee	Department	Manager
John	Finance	Smith
Mary	HR	Taylor
Susan	HR	Taylor
John	Sales	Smith

Are the ff. FDs satisfied by the above relation?

- {Department} → {Manager} YES! See red square above.
- {Employee} → {Department} NO! See green square above.
- {Employee, Manager} → {Department} NO!
  - ↳ As RC query, it's written as $\{() \mid \forall x, y, y', z \; R(x, y, z) \wedge R(x, y', z) \rightarrow y = y'\}$
  - Output can be {} (FALSE) or {()} (TRUE)
- {Manager} → {Department} NO!
  - ↳ $\{() \mid \forall x, y, z, x', y' \; R(x, y, z) \wedge R(x', y', z) \rightarrow y = y'\}$

→ **Key constraint** is a special case of FD $X \to Y$ where

Y is the whole set of attributes of a relation

↳ A key for a table is a set of attributes tht. uniquely identify a row

↳ A set of attributes X is a key for relation R if for every $t_1, t_2 \in R$

$$\pi_X(t_1) = \pi_X(t_2) \Rightarrow t_1 = t_2$$

i.e.

R

A	B	C
1	2	3
2	1	1
1	2	4

Let A be key for R, then $A \to A, B, C$.

The instance on the left does not satisfy the key constraint, bc. we have 2 rows that have the same value for A, but different values for B & C tgther.

2) INCLUSION DEPENDENCIES (IND)

→ are constraints of the form $R[X] \subseteq S[Y]$

where R, S are relations and X, Y are sequences of attributes

→ R and S satisfy $R[X] \subseteq S[Y]$ if

for every $t_1 \in R$ there exists $t_2 \in S$ such that $\pi_X(t_1) = \pi_Y(t_2)$

NOTE: The projection must respect the order of attributes [sequences]

→ INDs are referential constraints

↳ Link the contents of one table w/ the contents of another table

→ **Foreign key constraint** is the conjunction of 2 constraints:

1) An IND — $R[X] \subseteq S[Y]$

2) A key constraint — Y is key for S

ie

Employees	
Name	Dep
John	Finance
Mary	HR
John	HR
Linda	Finance
Susan	Sales

Departments	
Name	Mgr
Finance	John
HR	Mary
Sales	Linda

Which of the ff. INDs would the relation satisfy:

- Employees [Dep] ⊆ Departments [Name] YES!
  ↳ Employees must work for a department that is listed in Departments table
- Employees [Name] ⊆ Departments [Mgr] NO!
  ↳ Every employee is a manager; Susan is not a manager.
- Departments [Mgr] ⊆ Employees [Name] YES!
  ↳ Every manager is an employee
- Departments [Mgr, Name] ⊆ Employees [Name, Dep] NO!
  ↳ (Linda, Sales) is not in Employees table

ORDER is IMPORTANT!

BASIC SQL CONSTRAINTS

1) NOT NULL — to disallow null values

e.g.
```
CREATE TABLE Account (
    accnum   VARCHAR(12)  NOT NULL,
    branch   VARCHAR(30),
    custid   VARCHAR(10),
    balance  NUMERIC(14,2) DEFAULT 0);
```

The ff. insertion would fail:

```
INSERT INTO Account (branch, custid)
    VALUES ('London', 'cust1');
```

accnum & balance will take default value, but accnum has no default value hence it'll be NULL

2) UNIQUE — to declare keys

e.g.
```
CREATE TABLE Account (
    accnum   VARCHAR(12) UNIQUE,
    branch   VARCHAR(30),
    custid   VARCHAR(10),
    balance  NUMERIC(14,2));
```

ERROR (X)
```
INSERT INTO Account
VALUES (1, 'London', 'cust1', 100),
       (1, 'Edi', 'cust3', 200);
```

(√)
```
INSERT INTO Account
VALUES (NULL, 'London', 'cust1', 100),
       (NULL, 'Edi', 'cust3', 200);
```

UNIQUE allows null values and SQL doesn't consider it the same

↳ To declare compound keys, we use the ff. syntax:

```
CREATE TABLE Movies(
    m_title    VARCHAR(30),
    m_director VARCHAR(30),
    m_year     SMALLINT,
    UNIQUE (m_title, m_year));
```

This declares the set {m_title, m_year} as a key for Movies

↳ You can have repetitions in within each single column, but no repetitions when you consider two values together

3) PRIMARY KEY — UNIQUE + NOT NULL

```
CREATE TABLE Account (
    accnum VARCHAR(12) PRIMARY KEY,
    ⋮
```

Same as

```
CREATE TABLE Account (
    accnum VARCHAR(12) NOT NULL UNIQUE,
    ⋮
```

4) FOREIGN KEY – to reference attributes in other tables

```
CREATE TABLE <table1> (
    <attr> <type>,
    ...
    <attr> <type>,
    FOREIGN KEY (<list1>) REFERENCES <table2> (<list2>));
```

(<list1>): lists w/ same no. of attributes

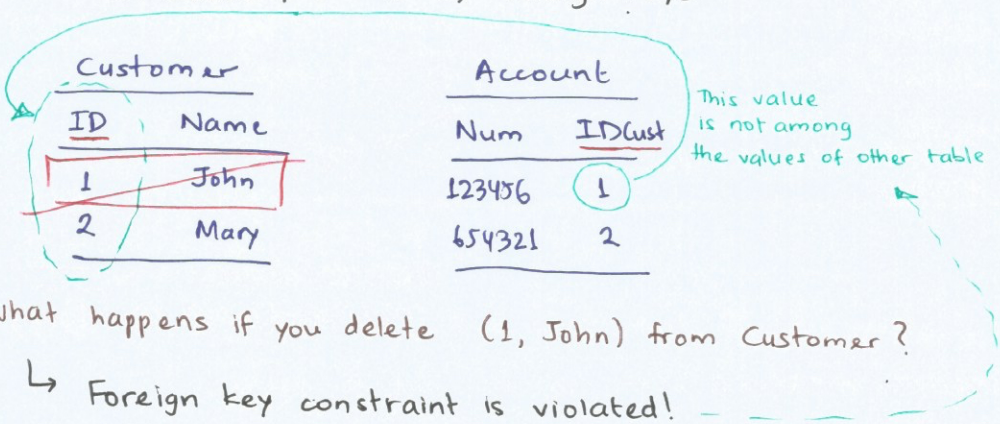
(<list2>): attributes here are unique in table2

↳ This does not enforce anything to be NOT NULL, this depends on the attributes tht. you are referencing (primary key / unique?)

REFERENTIAL INTEGRITY

→ Deletions can cause problems w/ foreign keys

i.e.



What happens if you delete (1, John) from Customer?

↳ Foreign key constraint is violated!

→ 3 ways to deal w/ this:

1) Reject the deletion operation

2) Propagate deletion into Account by deleting also (123456, 1)

↳ FOREIGN KEY... REFERENCES... ON DELETE CASCADE

3) "Don't know" approach

↳ keep tuple in Account but set the IDCust value to NULL

↳ ON DELETE SET NULL