CONSTRAINTS

CONSTRAINTS

THERE ARE COMMON DIFFERENT TYPES OF CONSTRAINTS

- ✓ NOT NULL CONSTRAINTS
- ✓ PRIMARY KEY CONSTRAINTS
- FOREIGN KEY CONSTRAINTS
- CHECK CONSTRAINTS

FOREIGN KEY CONSTRAINTS

THIS IS A TABLE NAMED 'CAMPUS_HOUSING'

StudentID	DormitoryName	AptNumber
1	Gandhi House	110
2	Akbar Hall	231
3	Gandhi House	345
4	NULL	NULL

COLUMNS ARE NAMED 'STUDENTID', 'DORMITORYNAME', 'APTNUMBER'

THIS IS A TABLE NAMED 'STUDENTS'

StudentID	FirstName	LastName	Gender	Email
1	Janani	Ravi	F	janani@loonycorn.com
2	Swetha	Kolalapudi	F	swetha@loonycorn.com
3	Navdeep	Singh	M	navdeep@loonycorn.com
4	Vitthal	Srinivasan	M	vitthal@loonycorn.com

COLUMNS ARE NAMED 'STUDENTID', 'FIRSTNAME', 'LASTNAME', 'GENDER' AND 'EMAIL'

FOREIGN KEY CONSTRAINTS

THE TWO TABLES ARE LINKED VIA THE COLUMN STUDENTID

StudentID	DormitoryName	AptNumber
1	Gandhi House	110
2	Akbar Hall	231
3	Gandhi House	345
4	NULL	NULL

StudentID	FirstName	LastName	Gender	Email
1	Janani	Ravi	F	janani@loonycorn.com
2	Swetha	Kolalapudi	F	swetha@loonycorn.com
3	Navdeep	Singh	M	navdeep@loonycorn.com
4	Vitthal	Srinivasan	M	vitthal@loonycorn.com

FOREIGN KEY CONSTRAINTS

THE TWO TABLES ARE LINKED VIA THE COLUMN STUDENTID

StudentID	DormitoryNa	AptNumber
1	Gandhi House	110
2	Akbar Hall	231
3	Gandhi House	345
4	NULL	NULL

StudentID	FirstName	LastName	Gender	Email
1	Janani	Ravi	F	janani@loonycorn.com
2	Swetha	Kolalapudi	F	swetha@loonycorn.com
3	Navdeep	Singh	M	navdeep@loonycorn.com
4	Vitthal	Srinivasan	M	vitthal@loonycorn.com

WHAT EXACTLY IS THE LINK BETWEEN THESE TWO TABLES?

WHAT EXACTLY IS THE LINK BETWEEN THESE TWO TABLES?

THE TABLE **STUDENTS** CONTAINS ONE ROW PER STUDENT.

THE TABLE **CAMPUS_HOUSING** CONTAINS ONE ROW FOR THE HOUSING INFORMATION OF EACH STUDENT

WHAT EXACTLY IS THE LINK BETWEEN THESE TWO TABLES?

THE TABLE STUDENTS CONTAINS ONE ROW PER STUDENT.

THE TABLE CAMPUS_HOUSING CONTAINS ONE ROW FOR THE HOUSING INFORMATION OF EACH STUDENT

THE CAMPUS_HOUSING TABLE SHOULD ONLY CONTAIN ROWS FOR STUDENTS WHO EXIST IN THE STUDENTS TABLE

THE CAMPUS_HOUSING TABLE SHOULD ONLY CONTAIN ROWS FOR STUDENTS WHO EXIST IN THE STUDENTS TABLE

WHY? BECAUSE THE CAMPUS_HOUSING TABLE ONLY MAKES SENSE WHEN VIEWED IN RELATION TO THE STUDENTS TABLE.

THE CAMPUS_HOUSING TABLE SHOULD ONLY CONTAIN ROWS FOR STUDENTS WHO EXIST IN THE STUDENTS TABLE

WHY? BECAUSE THE CAMPUS_HOUSING TABLE ONLY MAKES SENSE WHEN VIEWED IN RELATION TO THE STUDENTS TABLE.

THE STUDENTS TABLE WILL EXIST EVEN IF THE CAMPUS_HOUSING TABLE DOES NOT, BUT THE REVERSE IS NOT TRUE

THE CAMPUS_HOUSING TABLE SHOULD ONLY CONTAIN ROWS FOR STUDENTS WHO EXIST IN THE STUDENTS TABLE

WHY? BECAUSE THE CAMPUS_HOUSING TABLE ONLY MAKES SENSE WHEN VIEWED IN RELATION TO THE STUDENTS TABLE.

THE STUDENTS TABLE WILL EXIST EVEN IF THE CAMPUS_HOUSING TABLE DOES NOT, BUT THE REVERSE IS NOT TRUE

EACH VALUE IN THE STUDENTID COLUMN OF CAMPUS_HOUSING **MUST** ALREADY EXIST IN THE STUDENTID COLUMN OF THE STUDENTS TABLE

THE TWO TABLES ARE LINKED VIA THE COLUMN STUDENTID

StudentID	DormitoryNa	AptNumber
1	Gandhi House	110
2	Akbar Hall	231
3	Gandhi House	345
4	NULL	NULL

StudentID	FirstName	LastName	Gender	Email
1	Janani	Ravi	F	janani@loonycorn.com
2	Swetha	Kolalapudi	F	swetha@loonycorn.com
3	Navdeep	Singh	M	navdeep@loonycorn.com
4	Vitthal	Srinivasan	M	vitthal@loonycorn.com

EACH VALUE IN THE STUDENTID COLUMN OF CAMPUS_HOUSING MUST ALREADY EXIST IN THE STUDENTID COLUMN OF THE STUDENTS TABLE

THE TWO TABLES ARE LINKED VIA THE COLUMN STUDENTID

StudentID	DormitoryNa	AptNumber
1	Gandhi House	110
2	Akbar Hall	231
3	Gandhi House	345
4	NULL	NULL

StudentID	FirstName	LastName	Gender	Email
1	Janani	Ravi	F	janani@loonycorn.com
2	Swetha	Kolalapudi	F	swetha@loonycorn.com
3	Navdeep	Singh	M	navdeep@loonycorn.com
4	Vitthal	Srinivasan	M	vitthal@loonycorn.com

EACH VALUE IN THE STUDENTID COLUMN OF CAMPUS_HOUSING MUST ALREADY EXIST IN THE STUDENTID COLUMN OF THE STUDENTS TABLE

THE TWO TABLES ARE LINKED VIA THE COLUMN STUDENTID

StudentID	DormitoryNa	AptNumber
1	Gandhi House	110
2	Akbar Hall	231
3	Gandhi House	345
4	NULL	NULL

StudentID	FirstName	LastName	Gender	Email
1	Janani	Ravi	F	janani@loonycorn.com
2	Swetha	Kolalapudi	F	swetha@loonycorn.com
3	Navdeep	Singh	M	navdeep@loonycorn.com
4	Vitthal	Srinivasan	M	vitthal@loonycorn.com

EACH VALUE IN THE STUDENTID COLUMN OF CAMPUS_HOUSING MUST ALREADY EXIST IN THE STUDENTID COLUMN OF THE STUDENTS TABLE

THE TWO TABLES ARE LINKED VIA THE COLUMN STUDENTID

StudentID	DormitoryNa	AptNumber
1	Gandhi House	110
2	Akbar Hall	231
3	Gandhi House	345
4	NULL	NULL

StudentID	FirstName	LastName	Gender	Email
1	Janani	Ravi	F	janani@loonycorn.com
2	Swetha	Kolalapudi	F	swetha@loonycorn.com
3	Navdeep	Singh	M	navdeep@loonycorn.com
4	Vitthal	Srinivasan	M	vitthal@loonycorn.com

EACH VALUE IN THE STUDENTID COLUMN OF CAMPUS_HOUSING MUST ALREADY EXIST IN THE STUDENTID COLUMN OF THE STUDENTS TABLE

THE TWO TABLES ARE LINKED VIA THE COLUMN STUDENTID

StudentID	DormitoryNa	AptNumber
1	Gandhi House	110
2	Akbar Hall	231
3	Gandhi House	345
4	NULL	NULL

StudentID	FirstName	LastName	Gender	Email
1	Janani	Ravi	F	janani@loonycorn.com
2	Swetha	Kolalapudi	F	swetha@loonycorn.com
3	Navdeep	Singh	M	navdeep@loonycorn.com
4	Vitthal	Srinivasan	M	vitthal@loonycorn.com

EACH VALUE IN THE STUDENTID COLUMN OF CAMPUS_HOUSING MUST ALREADY EXIST IN THE STUDENTID COLUMN OF THE STUDENTS TABLE

THE TWO TABLES ARE LINKED VIA THE COLUMN STUDENTID

StudentID	DormitoryNa	AptNumber
1	Gandhi House	110
2	Akbar Hall	231
3	Gandhi House	345
4	NULL	NULL

StudentID	FirstName	LastName	Gender	Email
1	Janani	Ravi	F	janani@loonycorn.com
2	Swetha	Kolalapudi	F	swetha@loonycorn.com
3	Navdeep	Singh	M	navdeep@loonycorn.com
4	Vitthal	Srinivasan	M	vitthal@loonycorn.com

WE COULD ADD ROWS TO STUDENTS WITHOUT ADDING THEM TO CAMPUS_HOUSING..

WE COULD ADD ROWS TO STUDENTS WITHOUT ADDING THEM TO CAMPUS_HOUSING..

StudentID	DormitoryNa	AptNumber
1	Gandhi House	110
2	Akbar Hall	231
3	Gandhi House	345
4	NULL	NULL

AND THE CONTENTS OF THE 2 TABLES WOULD STILL MAKE SENSE..

StudentID	FirstName	LastName	Gender	Email
1	Janani	Ravi	F	janani@loonycorn.com
2	Swetha	Kolalapudi	F	swetha@loonycorn.com
3	Navdeep	Singh	M	navdeep@loonycorn.com
4	Vitthal	Srinivasan	M	vitthal@loonycorn.com
5	Ahmad	Mateen	NULL	mateenwa@gmail.com
99	Anu	Radha	NULL	anuradha@gmail.com
100	Pradeep	Shetty	NULL	pradeep@loonycorn.com

WE COULD ADD ROWS TO STUDENTS WITHOUT ADDING THEM TO CAMPUS_HOUSING..

AND THE CONTENTS OF THE 2 TABLES WOULD STILL MAKE SENSE..

MAKE SENSE IS NOT A TECHNICAL TERM - JUST COMMON SENSE!

BUT THIS ALSO HAS A TECHNICAL TERM (MORE IN A MINUTE)

WE COULD ADD ROWS TO STUDENTS WITHOUT ADDING THEM TO CAMPUS_HOUSING..

AND THE CONTENTS OF THE 2 TABLES WOULD STILL MAKE SENSE..

BUT IF WE ADDED ROWS TO CAMPUS_HOUSING WITHOUT ADDING THEM TO STUDENTS..

BUT IF WE ADDED ROWS TO CAMPUS_HOUSING WITHOUT ADDING THEM TO STUDENTS..

StudentID	DormitoryName	AptNumber
1	Gandhi House	110
2	Akbar Hall	231
3	Gandhi House	345
4	NULL	NULL
666	**The Bat Cave, Gotham**	**1007**

STUDENTID 666 DOES NOT APPEAR IN THE STUDENTS DATABASE!

StudentID	FirstName	LastName	Gender	Email
1	Janani	Ravi	F	janani@loonycorn.com
2	Swetha	Gnalapuri	F	swetha@loonycorn.com
3	Navdeep	Singh	M	navdeep@loonycorn.com
4	Vartha	Srinivasan	M	vartha@loonycorn.com
5	Ahmad	Mateen	NULL	mateenwa@gmail.com
99	Anu	Raghavan	NULL	anuraghavan@gmail.com
100	Pradeep	Shetty	NULL	pradeep@loonycorn.com

HOLY IMPOSSIBILITY!!

THERE'S A STUDENT MISSING FROM THE STUDENTS TABLE - AND IT MAYBE BATMAN!

BUT IF WE ADDED ROWS TO CAMPUS_HOUSING WITHOUT ADDING THEM TO STUDENTS..

StudentID	DormitoryName	ApiNumber
1	Gandhi House	10
2	Akbar Hall	231
3	Gandhi House	345
4	NULL	NULL
666	**The Bat Cave, Gotham**	**1007**

THE CONTENTS OF THE 2 TABLES WOULD NOT MAKE SENSE..

StudentID	FirstName	LastName	Gender	Email
1	Janani	Ravi	F	janani@loonycorn.com
2	Swetha	Kolalapudi	F	swetha@loonycorn.com
3	Navdeep	Singh	M	navdeep@loonycorn.com
4	Vitthal	Srinivasan	M	vitthal@loonycorn.com
5	Ahmad	Mateen	NULL	mateenwa@gmail.com
99	Anu	Radha	NULL	anuradha@gmail.com
100	Pradeep	Shetty	NULL	pradeep@loonycorn.com

BUT IF WE ADDED ROWS TO CAMPUS_HOUSING WITHOUT ADDING THEM TO STUDENTS..

THE CONTENTS OF THE 2 TABLES WOULD NOT MAKE SENSE..

AND THIS NEEDS TO BE EXPLICITLY SPECIFIED IN

THE TABLE SPECIFICATION OF THE CAMPUS_HOUSING TABLE

THE TABLE SPECIFICATION OF THE CAMPUS_HOUSING TABLE

THAT GETS US TO THE SQL CREATE TABLE STATEMENT FOR A TABLE LIKE THIS..

StudentID	DormitoryName	AptNumber

HOW DO WE CREATE TABLES?

THAT GETS US TO THE SQL CREATE TABLE STATEMENT FOR A TABLE LIKE THIS..

StudentID	DormitoryName	AptNumber

```
CREATE TABLE Campus_Housing
(
StudentID INT NOT NULL,
DormitoryName VARCHAR(50),
AptNumber INT,
CONSTRAINT fk_students_studentid
FOREIGN KEY (StudentID)
REFERENCES Students (StudentID)
)
```

HOW DO WE CREATE TABLES?

THAT GETS US TO THE SQL CREATE TABLE STATEMENT FOR A TABLE LIKE THIS..

StudentID	DormitoryName	AptNumber

```
CREATE TABLE Campus_Housing
 (
StudentID INT NOT NULL,
DormitoryName VARCHAR(50),
AptNumber INT,
CONSTRAINT fk_students_studentid
FOREIGN KEY (StudentID)
REFERENCES Students (StudentID)
 )
```

THIS BIT IS PRETTY STANDARD..

HOW DO WE CREATE TABLES?

THAT GETS US TO THE SQL CREATE TABLE STATEMENT FOR A TABLE LIKE THIS..

StudentID	DormitoryName	AptNumber

```
CREATE TABLE Campus_Housing
(
StudentID INT NOT NULL,
DormitoryName VARCHAR(50),
AptNumber INT,
CONSTRAINT fk_students_studentid
FOREIGN KEY (StudentID)
REFERENCES Students (StudentID)
)
```

AND SO ARE THE COLUMN SPECS

HOW DO WE CREATE TABLES?

THAT GETS US TO THE SQL CREATE TABLE STATEMENT FOR A TABLE LIKE THIS..

StudentID	DormitoryName	AptNumber

```
CREATE TABLE Campus_Housing
(
StudentID INT NOT NULL,
DormitoryName VARCHAR(50),
AptNumber INT,
CONSTRAINT fk_students_studentid
FOREIGN KEY (StudentID)
REFERENCES Students (StudentID)
)
```

NOTE THAT STUDENTID WON'T ACCEPT NULLS, BUT THE OTHER COLUMNS WILL

HOW DO WE CREATE TABLES?

THAT GETS US TO THE SQL CREATE TABLE STATEMENT FOR A TABLE LIKE THIS..

StudentID	DormitoryName	AptNumber

```
CREATE TABLE Campus_Housing
(
StudentID INT NOT NULL,
DormitoryName VARCHAR(50),
AptNumber INT,
CONSTRAINT fk_students_studentid
FOREIGN KEY (StudentID)
REFERENCES Students (StudentID)
)
```

BUT THIS IS WHERE IT GETS SUPER-INTERESTING

THIS LINE CREATES THE LINK BETWEEN CAMPUS_HOUSING AND STUDENTS

Campus_Housing

StudentID	DormitoryName	AptNumber

Students

StudentID	FirstName	LastName	Gender	Email

```
CREATE TABLE Campus_Housing
(
StudentID INT NOT NULL,
DormitoryName VARCHAR(50),
AptNumber INT,
CONSTRAINT fk_students_studentid
FOREIGN KEY (StudentID)
REFERENCES Students (StudentID)
)
```

THIS LINE CREATES THE LINK BETWEEN CAMPUS_HOUSING AND STUDENTS

Campus_Housing

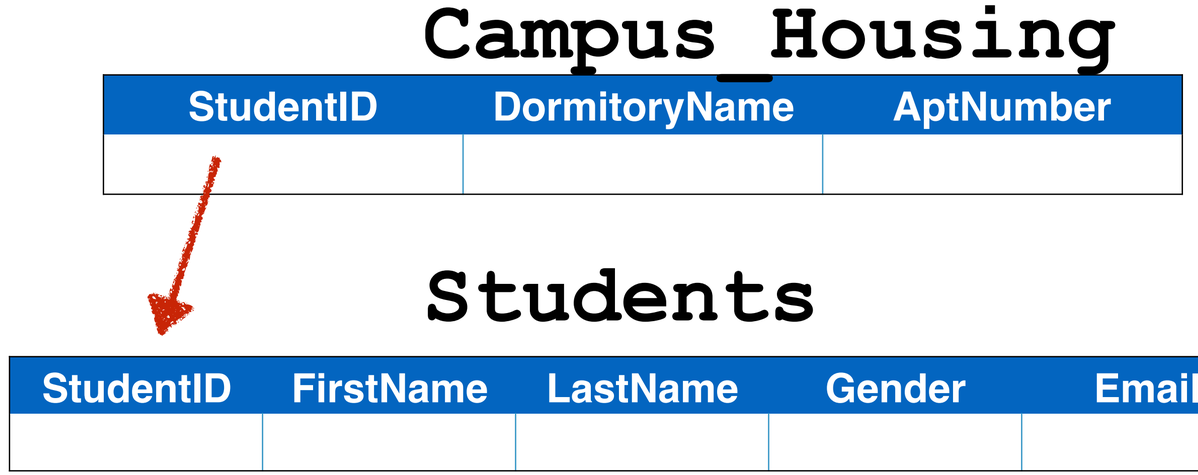
StudentID	DormitoryName	AptNumber

Students

StudentID	FirstName	LastName	Gender	Email

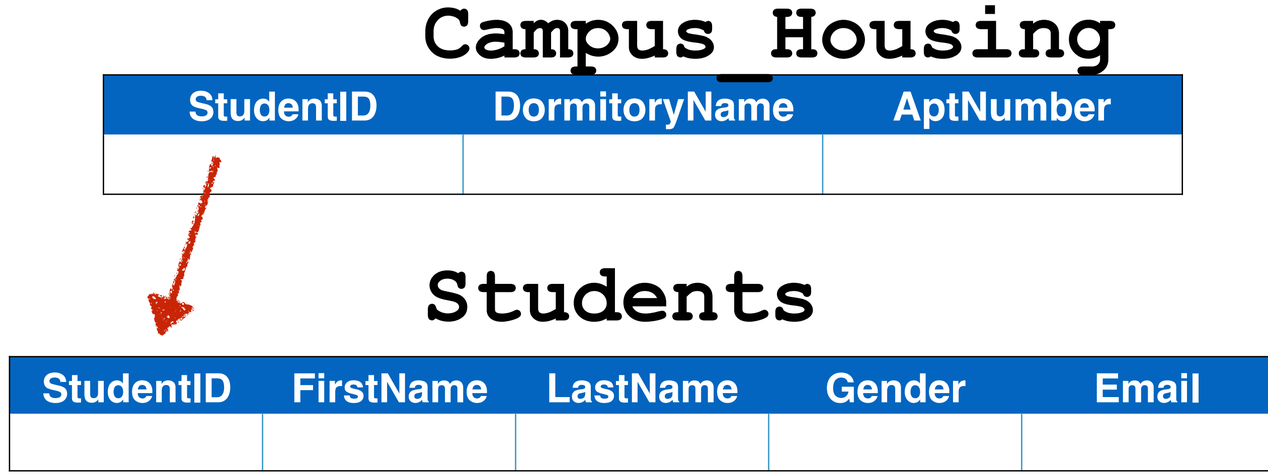
```
CREATE TABLE Campus_Housing
(
StudentID INT NOT NULL,
DormitoryName VARCHAR(50),
AptNumber INT,
CONSTRAINT fk_students_studentid
FOREIGN KEY (StudentID)
REFERENCES Students (StudentID)
)
```

THIS LINE CREATES A CONSTRAINT, THAT MUST BE SATISFIED BY THE TABLE CAMPUS_HOUSING




```
CREATE TABLE Campus_Housing
(
StudentID INT NOT NULL,
DormitoryName VARCHAR(50),
AptNumber INT,
CONSTRAINT fk_students_studentid
FOREIGN KEY (StudentID)
REFERENCES Students (StudentID)
)
```

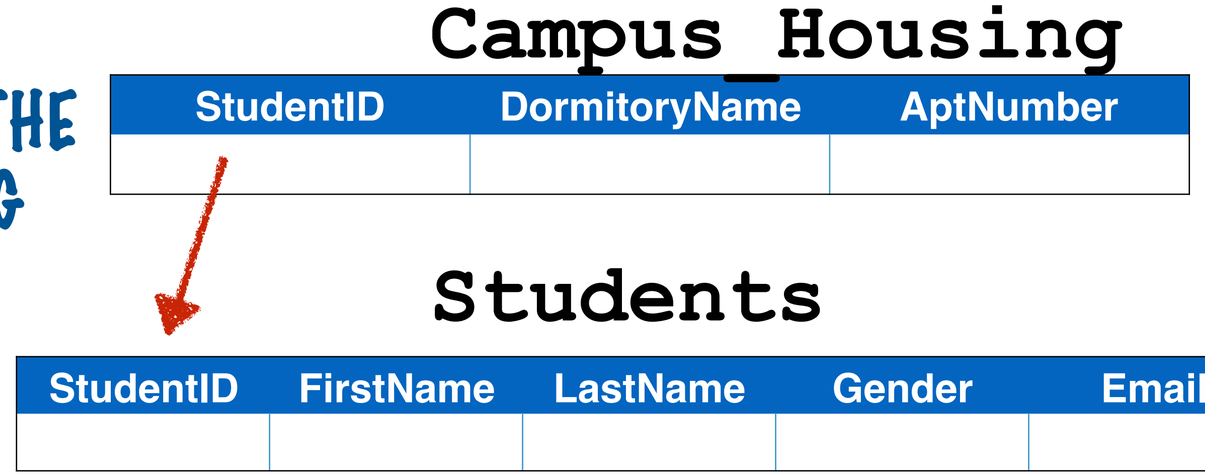
THIS LINE CREATES A CONSTRAINT, THAT MUST BE SATISFIED BY THE TABLE CAMPUS_HOUSING



THAT CONSTRAINT IS THAT: EACH VALUE IN THE STUDENTID COLUMN OF CAMPUS_HOUSING MUST ALREADY EXIST IN THE STUDENTID COLUMN OF THE STUDENTS TABLE

THIS LINE CREATES A CONSTRAINT, THAT MUST BE SATISFIED BY THE TABLE CAMPUS_HOUSING

THAT CONSTRAINT IS THAT: EACH VALUE IN THE STUDENTID COLUMN OF CAMPUS_HOUSING MUST ALREADY EXIST IN THE STUDENTID COLUMN OF THE STUDENTS TABLE



SUCH CONSTRAINTS ARE CALLED FOREIGN KEY CONSTRAINTS, OR REFERENTIAL INTEGRITY CONSTRAINTS

THIS LINE CREATES A CONSTRAINT, THAT MUST BE SATISFIED BY THE TABLE CAMPUS_HOUSING

THAT CONSTRAINT IS THAT: EACH VALUE IN THE STUDENTID COLUMN OF CAMPUS_HOUSING MUST ALREADY EXIST IN THE STUDENTID COLUMN OF THE STUDENTS TABLE

Campus_Housing

StudentID	DormitoryName	AptNumber

Students

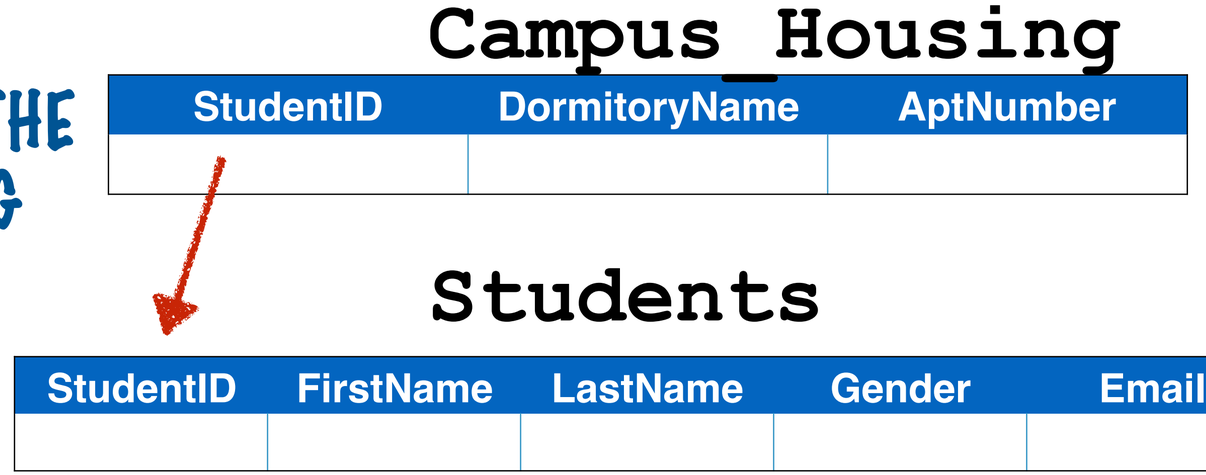
StudentID	FirstName	LastName	Gender	Email

SUCH CONSTRAINTS ARE CALLED FOREIGN KEY CONSTRAINTS, OR REFERENTIAL INTEGRITY CONSTRAINTS

"FOREIGN KEY" BECAUSE THE CONSTRAINT IS ABOUT A COLUMN THAT IS A KEY IN ANOTHER TABLE

THIS LINE CREATES A CONSTRAINT, THAT MUST BE SATISFIED BY THE TABLE CAMPUS_HOUSING

THAT CONSTRAINT IS THAT: EACH VALUE IN THE STUDENTID COLUMN OF CAMPUS_HOUSING MUST ALREADY EXIST IN THE STUDENTID COLUMN OF THE STUDENTS TABLE



SUCH CONSTRAINTS ARE CALLED FOREIGN KEY CONSTRAINTS, OR REFERENTIAL INTEGRITY CONSTRAINTS

"FOREIGN KEY" BECAUSE THE CONSTRAINT IS ABOUT A COLUMN THAT IS A KEY IN ANOTHER TABLE

"REFERENTIAL INTEGRITY" BECAUSE ONE TABLE REFERS TO ANOTHER TO CHECK ITS INTEGRITY

SUCH CONSTRAINTS ARE CALLED FOREIGN KEY CONSTRAINTS, OR REFERENTIAL INTEGRITY CONSTRAINTS

"FOREIGN KEY" BECAUSE THE CONSTRAINT IS ABOUT A COLUMN THAT IS A KEY IN ANOTHER TABLE

Campus_Housing

StudentID	DormitoryName	AptNumber

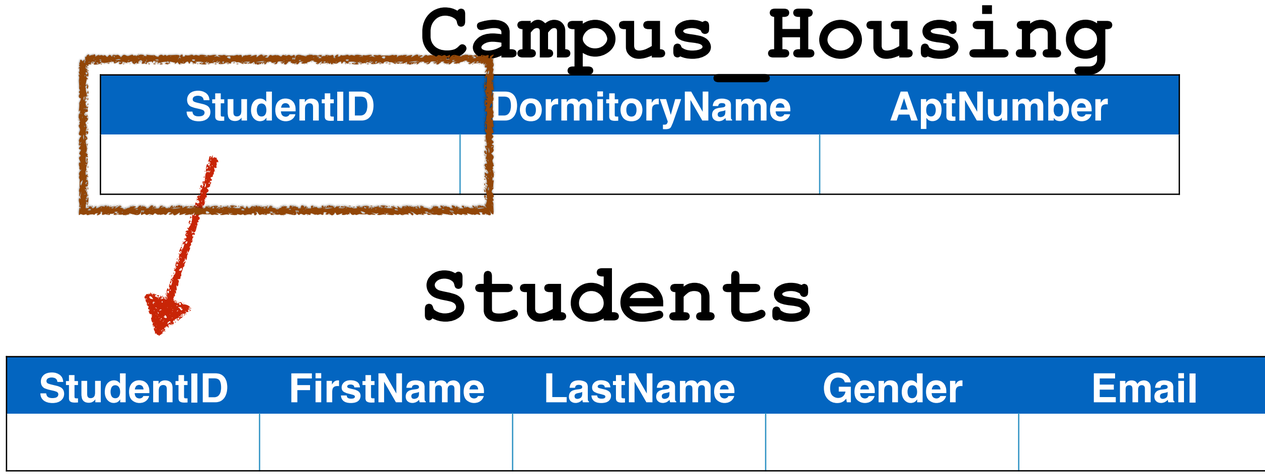
Students

StudentID	FirstName	LastName	Gender	Email

`Students.StudentID` is a key in Students and referenced by
`Campus_Housing.StudentID` to check if its integrity is maintained

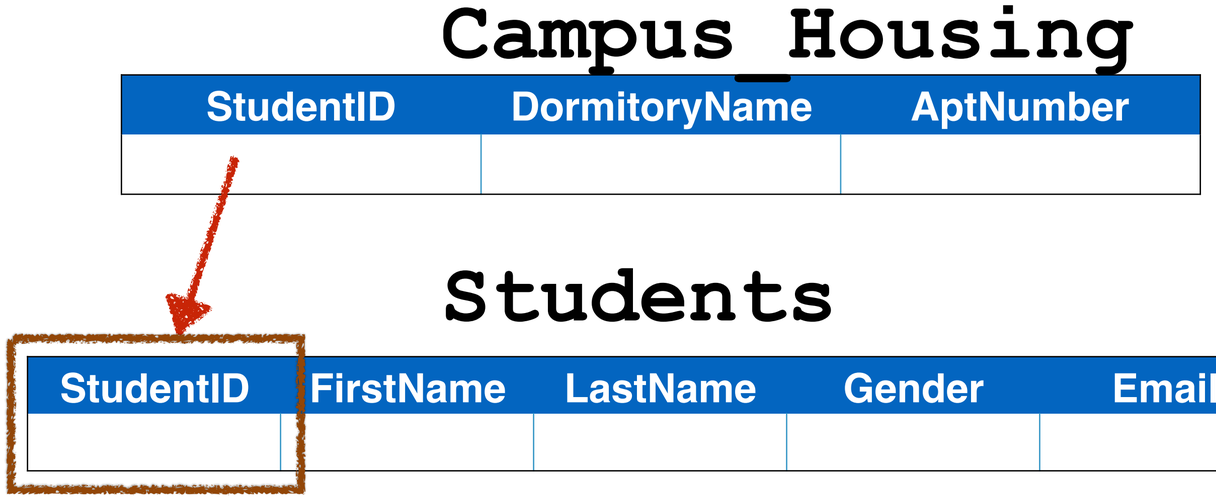
"REFERENTIAL INTEGRITY" BECAUSE ONE TABLE REFERS TO ANOTHER TO CHECK ITS INTEGRITY

Students.StudentID is a key in Students and referenced by Campus_Housing.StudentID to check if its integrity is maintained




```
CREATE TABLE Campus_Housing
(
StudentID INT NOT NULL,
DormitoryName VARCHAR(50),
AptNumber INT,
CONSTRAINT fk_students_studentid
FOREIGN KEY (StudentID)
REFERENCES Students (StudentID)
)
```

Students.StudentID is a key in Students and referenced by Campus_Housing.StudentID to check if its integrity is maintained




```
CREATE TABLE Campus_Housing
(
StudentID INT NOT NULL,
DormitoryName VARCHAR(50),
AptNumber INT,
CONSTRAINT fk_students_studentid
FOREIGN KEY (StudentID)
REFERENCES Students (StudentID)
)
```

Students.StudentID is a key in Students and referenced by Campus_Housing.StudentID to check if its integrity is maintained

Campus_Housing

StudentID	DormitoryName	AptNumber

Students

StudentID	FirstName	LastName	Gender	Email

```
CREATE TABLE Campus_Housing
(
StudentID INT NOT NULL,
DormitoryName VARCHAR(50),
AptNumber INT,
CONSTRAINT fk_students_studentid
FOREIGN KEY (StudentID)
REFERENCES Students (StudentID)
)
```

DELETE AND UPDATES MAKE FOREIGN KEY CONSTRAINTS QUITE COMPLICATED

DELETE AND UPDATES MAKE FOREIGN KEY CONSTRAINTS QUITE COMPLICATED

SAY WE HAVE 2 TABLES, FOR CURRENT_EMPLOYEES, AND UPCOMING_PROMOTIONS

Upcoming_Promotions

EmployeeID	Current Title	New Title

Current_Employees

EmployeeID	FirstName	LastName

DELETE AND UPDATES MAKE FOREIGN KEY CONSTRAINTS QUITE COMPLICATED

SAY WE HAVE 2 TABLES, FOR CURRENT_EMPLOYEES, AND UPCOMING_PROMOTIONS

Upcoming_Promotions

EmployeeID	Current Title	New Title

Current_Employees

EmployeeID	FirstName	LastName

CLEARLY, ONLY CURRENT EMPLOYEES CAN BE PROMOTED - I.E. A FOREIGN KEY CONSTRAINT IS CALLED FOR

SAY WE HAVE 2 TABLES, FOR CURRENT_EMPLOYEES, AND UPCOMING_PROMOTIONS

CLEARLY, ONLY CURRENT EMPLOYEES CAN BE PROMOTED - I.E. A FOREIGN KEY CONSTRAINT IS CALLED FOR

Upcoming_Promotions

EmployeeID	Current Title	New Title

```
CREATE TABLE Upcoming_Promotions
(
EmployeeID INT NOT NULL,
CurrentTitle VARCHAR(50),
NewTitle VARCHAR(50),
CONSTRAINT fk_upcomingpromo_current
FOREIGN KEY (EmployeeID)
REFERENCES Current_Employees (EmployeeID)
)
```

Current_Employees

EmployeeID	FirstName	LastName

```
CREATE TABLE Current_Employees
(
EmployeeID INT NOT NULL PRIMARY KEY,
FirstName VARCHAR(50) NOT NULL,
LastName VARCHAR(50) NOT NULL,
)
```

SAY WE HAVE 2 TABLES, FOR CURRENT_EMPLOYEES, AND UPCOMING_PROMOTIONS

CLEARLY, ONLY CURRENT EMPLOYEES CAN BE PROMOTED - I.E. A FOREIGN KEY CONSTRAINT IS CALLED FOR

Upcoming_Promotions

EmployeeID	Current Title	New Title

```
CREATE TABLE Upcoming_Promotions
(
EmployeeID INT NOT NULL,
CurrentTitle VARCHAR(50),
NewTitle VARCHAR(50),
CONSTRAINT fk_upcomingpromo_current
FOREIGN KEY (EmployeeID)
REFERENCES Current_Employees (EmployeeID)
)
```

Current_Employees

EmployeeID	FirstName	LastName

```
CREATE TABLE Current_Employees
(
EmployeeID INT NOT NULL PRIMARY KEY,
FirstName VARCHAR(50) NOT NULL,
LastName VARCHAR(50) NOT NULL,
)
```

SAY WE HAVE 2 TABLES, FOR CURRENT_EMPLOYEES, AND UPCOMING_PROMOTIONS

CLEARLY, ONLY CURRENT EMPLOYEES CAN BE PROMOTED - I.E. A FOREIGN KEY CONSTRAINT IS CALLED FOR

Upcoming_Promotions

EmployeeID	Current Title	New Title

Current_Employees

EmployeeID	FirstName	LastName

```
CREATE TABLE Upcoming_Promotions
(
EmployeeID INT NOT NULL,
CurrentTitle VARCHAR(50),
NewTitle VARCHAR(50),
CONSTRAINT fk_upcomingpromo_current
FOREIGN KEY (EmployeeID)
REFERENCES Current_Employees (EmployeeID)
)
```

```
CREATE TABLE Current_Employees
(
EmployeeID INT NOT NULL PRIMARY KEY,
FirstName VARCHAR(50) NOT NULL,
LastName VARCHAR(50) NOT NULL,
)
```

EmployeeID	Current Title	New Title
346	Senior Slide Monkey	Chief Presentation Officer
397	Director, Schmoozing	Senior Director, Schmoozing

EmployeeID	FirstName	LastName
346	John	Doe
438	James	Bull
349	Jack	Daw
397	Tim	Shaw

BREAKING NEWS! EMPLOYEE #346 GOT A BETTER GIG ELSEWHERE, AND QUIT WITHOUT WAITING FOR HIS PROMOTION.

ERRM..WHAT NOW FOR THE UPCOMING_PROMOTIONS TABLE?

BREAKING NEWS! EMPLOYEE #346 GOT A BETTER GIG ELSEWHERE, AND QUIT WITHOUT WAITING FOR HIS PROMOTION.

ERRM..WHAT NOW FOR THE UPCOMING_PROMOTIONS TABLE?

Upcoming_Promotions

EmployeeID	Current Title	New Title

```
CREATE TABLE Upcoming_Promotions
(
EmployeeID INT NOT NULL,
CurrentTitle VARCHAR(50),
NewTitle VARCHAR(50),
CONSTRAINT fk_upcomingpromo_current
FOREIGN KEY (EmployeeID)
REFERENCES Current_Employees (EmployeeID)
)
```

EmployeeID	Current Title	New Title
346	Senior Slide Monkey	Chief Presentation Officer
397	Director, Schmoozing	Senior Director, Schmoozing

Current_Employees

EmployeeID	FirstName	LastName

```
CREATE TABLE Current_Employees
(
EmployeeID INT NOT NULL PRIMARY KEY,
FirstName VARCHAR(50) NOT NULL,
LastName VARCHAR(50) NOT NULL,
)
```

EmployeeID	FirstName	LastName
346	John	Doe
438	James	Bull
349	Jack	Daw
397	Tim	Shaw

BREAKING NEWS! EMPLOYEE #346 GOT A BETTER GIG ELSEWHERE, AND QUIT WITHOUT WAITING FOR HIS PROMOTION.

ERRM..WHAT NOW FOR THE UPCOMING_PROMOTIONS TABLE?

Upcoming_Promotions

EmployeeID	Current Title	New Title
346	Senior Slide Monkey	Chief Presentation Officer
397	Director, Schmoozing	Senior Director, Schmoozing

Current_Employees

EmployeeID	FirstName	LastName
346	John	Doe
438	James	Bull
349	Jack	Daw
397	Tim	Shaw

WE HAVE TO DELETE 346 FROM THE CURRENT_EMPLOYEES TABLE - WHICH MEANS WE HAVE TO DECIDE WHAT TO DO ABOUT 346'S UPCOMING PROMOTION

BREAKING NEWS! EMPLOYEE #346 GOT A BETTER GIG ELSEWHERE, AND QUIT WITHOUT WAITING FOR HIS PROMOTION.

ERRM..WHAT NOW FOR THE UPCOMING_PROMOTIONS TABLE?

Upcoming_Promotions

EmployeeID	Current Title	New Title
346	Senior Slide Monkey	Chief Presentation Officer
397	Director, Schmoozing	Senior Director, Schmoozing

Current_Employees

EmployeeID	FirstName	LastName
346	John	Doe
438	James	Bull
349	Jack	Daw
397	Tim	Shaw

WE HAVE TO DELETE 346 FROM THE CURRENT_EMPLOYEES TABLE - WHICH MEANS WE HAVE TO DECIDE WHAT TO DO ABOUT 346'S UPCOMING PROMOTION

OPTION 1: CASCADE THE DELETION AND ZAP FROM UPCOMING_PROMOTIONS TOO

BREAKING NEWS! EMPLOYEE #346 GOT A BETTER GIG ELSEWHERE, AND QUIT WITHOUT WAITING FOR HIS PROMOTION.

ERRM..WHAT NOW FOR THE UPCOMING_PROMOTIONS TABLE?

Upcoming_Promotions

EmployeeID	Current Title	New Title
397	Director, Schmoozing	Senior Director, Schmoozing

Current_Employees

EmployeeID	FirstName	LastName
438	James	Bull
349	Jack	Daw
397	Tim	Shaw

WE HAVE TO DELETE 346 FROM THE CURRENT_EMPLOYEES TABLE - WHICH MEANS WE HAVE TO DECIDE WHAT TO DO ABOUT 346'S UPCOMING PROMOTION

OPTION 1: CASCADE THE DELETION AND ZAP FROM UPCOMING_PROMOTIONS TOO

OPTION 1: CASCADE THE DELETION AND ZAP FROM `UPCOMING_PROMOTIONS` TOO

THIS IS CALLED "ON-DELETE-CASCADE"

OPTION 1: CASCADE THE DELETION AND ZAP FROM UPCOMING_PROMOTIONS TOO

THIS IS CALLED "ON-DELETE-CASCADE"

```
CREATE TABLE Upcoming_Promotions
(
EmployeeID INT NOT NULL,
CurrentTitle VARCHAR(50),
NewTitle VARCHAR(50),
CONSTRAINT fk_upcomingpromo_current
FOREIGN KEY (EmployeeID)
REFERENCES Current_Employees (EmployeeID)
ON DELETE CASCADE
)
```

BREAKING NEWS! EMPLOYEE #346 GOT A BETTER GIG ELSEWHERE, AND QUIT WITHOUT WAITING FOR HIS PROMOTION.

ERRM..WHAT NOW FOR THE UPCOMING_PROMOTIONS TABLE?

Upcoming_Promotions

EmployeeID	Current Title	New Title
346	Senior Slide Monkey	Chief Presentation Officer
397	Director, Schmoozing	Senior Director, Schmoozing

Current_Employees

EmployeeID	FirstName	LastName
346	John	Doe
438	James	Bull
349	Jack	Daw
397	Tim	Shaw

WE HAVE TO DELETE 346 FROM THE CURRENT_EMPLOYEES TABLE - WHICH MEANS WE HAVE TO DECIDE WHAT TO DO ABOUT 346'S UPCOMING PROMOTION

OPTION 2: DO NOTHING IN UPCOMING_PROMOTIONS AT ALL

BREAKING NEWS! EMPLOYEE #346 GOT A BETTER GIG ELSEWHERE, AND QUIT WITHOUT WAITING FOR HIS PROMOTION.

ERRM..WHAT NOW FOR THE UPCOMING_PROMOTIONS TABLE?

Upcoming_Promotions

EmployeeID	Current Title	New Title
346	Senior Slide Monkey	Chief Presentation Officer
397	Director, Schmoozing	Senior Director, Schmoozing

ORPHAN TUPLE!!

Current_Employees

EmployeeID	FirstName	LastName
438	James	Bull
349	Jack	Daw
397	Tim	Shaw

WE HAVE TO DELETE 346 FROM THE CURRENT_EMPLOYEES TABLE - WHICH MEANS WE HAVE TO DECIDE WHAT TO DO ABOUT 346'S UPCOMING PROMOTION

OPTION 2: DO NOTHING IN UPCOMING_PROMOTIONS AT ALL

OPTION 2: DO NOTHING IN UPCOMING_PROMOTIONS AT ALL

THIS IS CALLED "ON-DELETE-DO NOTHING", AND IT LEAVES AN ORPHAN TUPLE IN THE CHILD TABLE :-(

OPTION 2: DO NOTHING IN UPCOMING_PROMOTIONS AT ALL

THIS IS CALLED "ON-DELETE-DO NOTHING", AND IT LEAVES AN ORPHAN TUPLE IN THE CHILD TABLE :-(

```
CREATE TABLE Upcoming_Promotions
(
EmployeeID INT NOT NULL,
CurrentTitle VARCHAR(50),
NewTitle VARCHAR(50),
CONSTRAINT fk_upcomingpromo_current
FOREIGN KEY (EmployeeID)
REFERENCES Current_Employees (EmployeeID)
ON DELETE NO ACTION
)
```

BREAKING NEWS! EMPLOYEE #346 GOT A BETTER GIG ELSEWHERE, AND QUIT WITHOUT WAITING FOR HIS PROMOTION.

ERRM..WHAT NOW FOR THE UPCOMING_PROMOTIONS TABLE?

Upcoming_Promotions

EmployeeID	Current Title	New Title
346	Senior Slide Monkey	Chief Presentation Officer
397	Director, Schmoozing	Senior Director, Schmoozing

Current_Employees

EmployeeID	FirstName	LastName
346	John	Doe
438	James	Bull
349	Jack	Daw
397	Tim	Shaw

WE HAVE TO DELETE 346 FROM THE `CURRENT_EMPLOYEES` TABLE - WHICH MEANS WE HAVE TO DECIDE WHAT TO DO ABOUT 346'S UPCOMING PROMOTION

OPTION 3: SET THE CORRESPONDING VALUE IN `UPCOMING_PROMOTIONS` TO NULL

BREAKING NEWS! EMPLOYEE #346 GOT A BETTER GIG ELSEWHERE, AND QUIT WITHOUT WAITING FOR HIS PROMOTION.

ERRM..WHAT NOW FOR THE UPCOMING_PROMOTIONS TABLE?

Upcoming_Promotions

EmployeeID	Current Title	New Title
NULL	Senior Slide Monkey	Chief Presentation Officer
397	Director, Schmoozing	Senior Director, Schmoozing

NOT-NULL VIOLATED!

Current_Employees

EmployeeID	FirstName	LastName
438	James	Bull
349	Jack	Daw
397	Tim	Shaw

WE HAVE TO DELETE 346 FROM THE CURRENT_EMPLOYEES TABLE - WHICH MEANS WE HAVE TO DECIDE WHAT TO DO ABOUT 346'S UPCOMING PROMOTION

OPTION 3: SET THE CORRESPONDING VALUE IN UPCOMING_PROMOTIONS TO NULL

OPTION 3: SET THE CORRESPONDING VALUE IN `UPCOMING_PROMOTIONS` TO NULL

THIS IS CALLED "ON-DELETE-SET TO NULL", AND IT LEAVES A NULL VALUE IN A NOT NULL COLUMN IN THE CHILD TABLE :-(

OPTION 3: SET THE CORRESPONDING VALUE IN UPCOMING_PROMOTIONS TO NULL

THIS IS CALLED "ON-DELETE-SET TO NULL", AND IT LEAVES A NULL VALUE IN A NOT NULL COLUMN IN THE CHILD TABLE :-(

```
CREATE TABLE Upcoming_Promotions
(
EmployeeID INT NOT NULL,
CurrentTitle VARCHAR(50),
NewTitle VARCHAR(50),
CONSTRAINT fk_upcomingpromo_current
FOREIGN KEY (EmployeeID)
REFERENCES Current Employees (EmployeeID)
ON DELETE SET NULL
)
```

THERE ARE 'ON UPDATE CASCADE' RULES
EXACTLY LIKE 'ON DELETE CASCADE' RULES

THERE ARE 'ON UPDATE CASCADE' RULES EXACTLY LIKE 'ON DELETE CASCADE' RULES

```
CREATE TABLE Upcoming_Promotions
(
EmployeeID INT NOT NULL,
CurrentTitle VARCHAR(50),
NewTitle VARCHAR(50),
CONSTRAINT fk_upcomingpromo_current
FOREIGN KEY (EmployeeID)
REFERENCES Current_Employees (EmployeeID)
ON DELETE CASCADE
ON UPDATE CASCADE
)
```

CASCADING IS THE 'BEST' OPTION, BUT IT MAKES UPDATES/DELETES VERY SLOW

CASCADING IS THE 'BEST' OPTION, BUT IT MAKES UPDATES/DELETES VERY SLOW

THE OTHER OPTIONS ARE QUICKER, BUT LEAVE THE DATABASE IN AN INCONSISTENT STATE

DELETE AND UPDATES MAKE FOREIGN KEY CONSTRAINTS QUITE COMPLICATED

CONSTRAINTS

THERE ARE COMMON DIFFERENT TYPES OF CONSTRAINTS

- ✓ NOT NULL CONSTRAINTS
- ✓ PRIMARY KEY CONSTRAINTS
- FOREIGN KEY CONSTRAINTS
- CHECK CONSTRAINTS