CS2102: Database Systems

SQL (Part 1)

Announcement

Announcement

Project Group

- Deadline for group formation is this ***Saturday***, **28 January 2023** by 12:00 Noon
  - Groups with fewer than 4 members may be *merged* or *allocated* a random member

Tutorial

- Starts this **Wednesday**
- Please prepare the material before hand
- Solutions will be released on Friday evening *(18:00)*

Catch-Up Session

- **Session I:** Friday, 27 January 2023 at 18:00
  - Q&A Session
  - Will be recorded

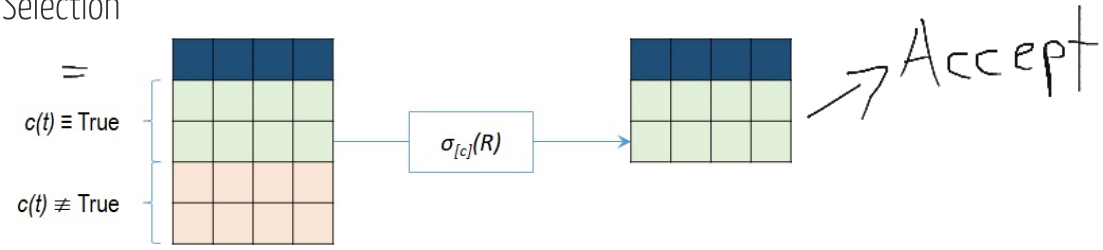
Recap

Recap

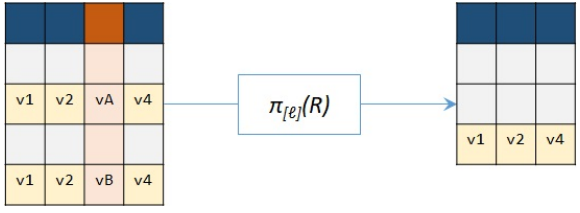
Unary
Set
Join

Unary

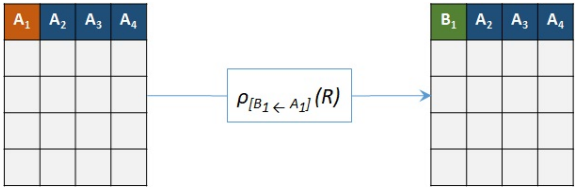
Selection



Projection



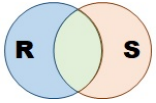
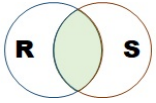
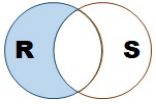
Renaming



Recap

Unary
Set
Join

Set

Operation	Operator	Visualization	Description
Union	R ∪ S		A relation containing all tuples that are in *R* or *S*
Intersection	R ∩ S		A relation containing all tuples that are in *R* and *S*
Difference	R – S		A relation containing all tuples that are in *R* but not in *S*

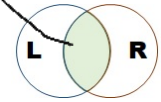
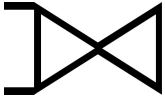
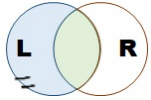
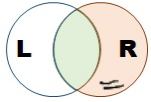
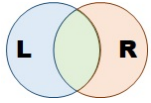
Union - Compatible

Recap

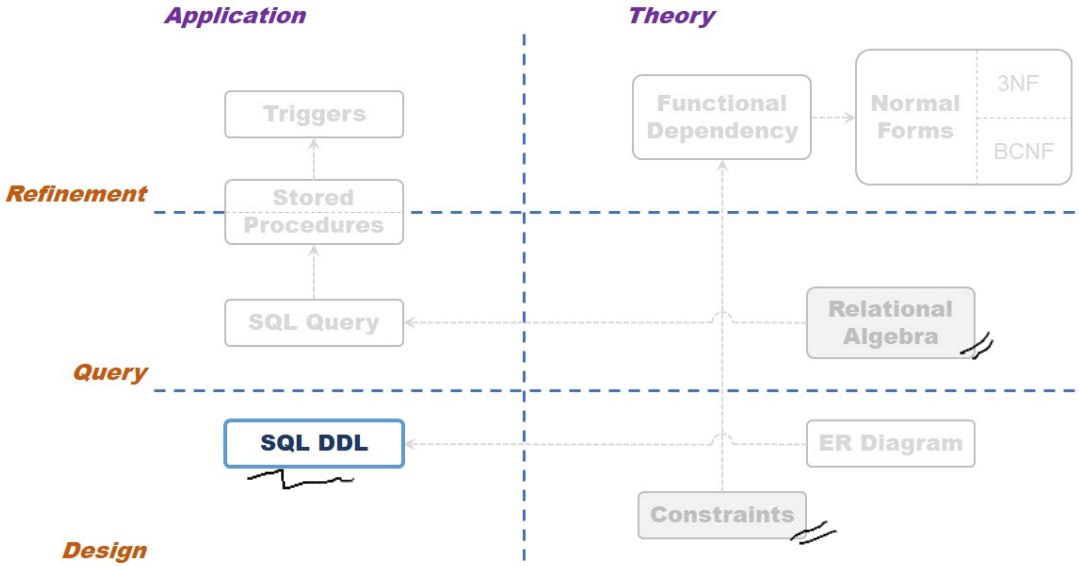
Unary
Set
Join

Joins

satisfy condition

L	Operator	R	Visualization
			
Keep dangling			
		Keep dangling	
Keep dangling		Keep dangling	

Roadmap


Application
Theory
Triggers
Functional Dependency
Normal Forms
3NF
BCNF
Refinement
Stored Procedures
SQL Query
Relational Algebra
Query
SQL DDL
ER Diagram
Constraints
Design

Roadmap

Overview

Overview

SQL Overview

- History and Usages

Creating/Insert/Deleting

- Basic DDL
- Basic DML

Integrity Constraints

- Unique and Keys
- General

Modifying Database

- Alter and Drop

Deferrable Constraints

SQL Overview

SQL Overview

SQL: Structured Query Language

What Is It?

- De-facto standard language to "talk" to RDBMS
  - Developed by Donald D. Chamberlin and Raymond F. Boyce *(IBM Research, 1974)*
  - Originally called *SEQUEL* *(Structured English Query Language)*
- SQL is a **domain-specific language** *(i.e., not a general-purpose like Python)*
- SQL is a **declarative language**
  - Focus on ***what*** to compute, instead of ***how*** to compute
  - Somewhat opposite to relational algebra expression *(which is **imperative**)*
- Based on *multi-set*/*bag* *(may contain duplicate rows)* instead of *set*

SQL Standard

- First standard is SQL-86; the most recent standard is SQL-2019 *(new standard every ~3-5 years)*
- New standards introduce new language concepts *(e.g., support new features of RDBMS)*
- Many RDBMS add their own "*flavor*" to SQL

SQL Overview

Using SQL

Non-Interactive SQL

> Included in application written in a host language

Statement Level Interface (SLI)

- Application is a *mixture* of host language *(e.g., C, Java, Python)* and SQL statements
  - ***Example:*** Embedded SQL, Dynamic SQL

Call Level Interface (CLI)

- Application is *entirely* written in the host language *(e.g., C, Java, Python)*
  - ***Example:*** ODBC *(Open DataBase Connectivity)*, JDBC *(Java DataBase Connectivity)*

SQL Overview

Using SQL

Interactive SQL

- Directly writing SQL statements to an interface

Command Line

psql

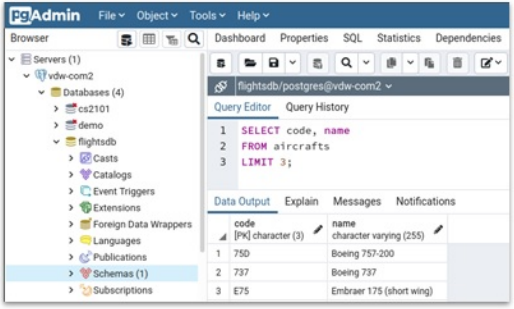
```
          List of relations
 Schema |    Name     | Type  |  Owner
--------+-------------+-------+----------
 public | aircrafts   | table | postgres
 public | airports    | table | postgres
 public | countries   | table | postgres
 public | flightcodes | table | postgres
 public | flights     | table | postgres
(5 rows)

flightsdb=# SELECT code, name FROM aircrafts LIMIT 3;
 code |           name
------+--------------------------
 75D  | Boeing 757-200
 737  | Boeing 737
 E75  | Embraer 175 (short wing)
(3 rows)

flightsdb=#
```

Graphical User Interface

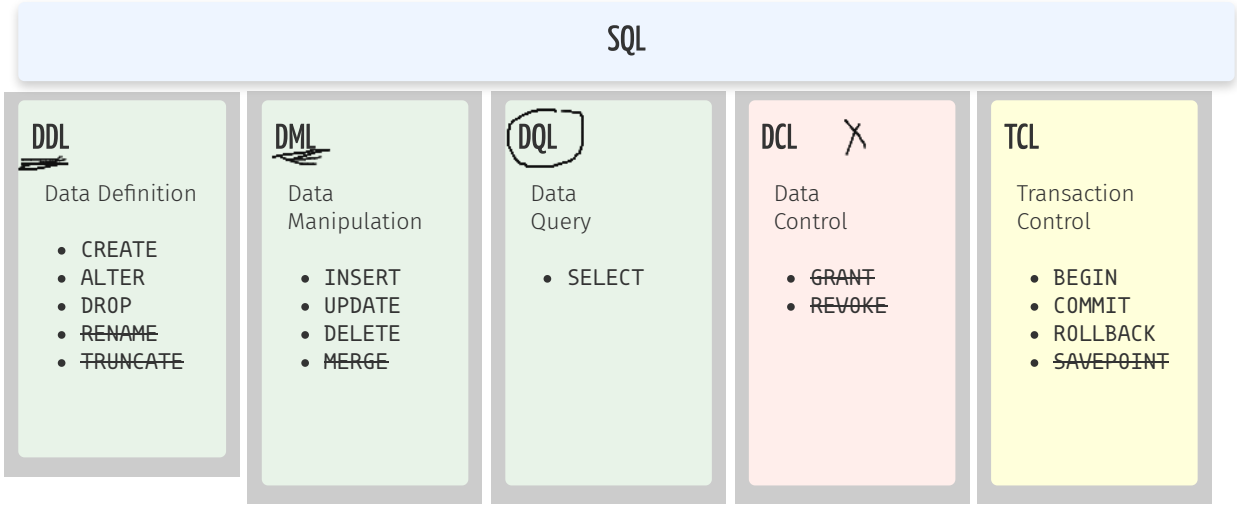
pgAdmin



- This will be our preferred mode

SQL Overview

Types of Commands/Statements



Creating/Insert/Deleting

Creating/Insert/Deleting

CREATE TABLE

Syntax

```
CREATE TABLE <table_name> (
  <attr1> <type1> [<column_constraint>],
  <attr2> <type2> [<column_constraint>],
    ⋮
  <attrn> <typen> [<column_constraint>],
  [<table_constraint>], -- comment
  [<table_constraint>], /* comment */
    ⋮
  [<table_constraint>]  -- no comma
);
```

Notes

- SQL is ***case-insensitive***
  - You can use *uppercase* or *lowercase*
- ***Convention:***
  - Keywords should be in uppercase
  - Try to align elements

cAn MiX uPpeR AnD LoWEr

Creating/Insert/Deleting

CREATE
- *Syntax*
- ***Semantics***
- *Example*

Data Types
INSERT
DELETE

CREATE TABLE

Syntax

```
CREATE TABLE <table_name> (
  <attr1> <type1> [<column_constraint>],
  <attr2> <type2> [<column_constraint>],
    ⋮
  <attrn> <typen> [<column_constraint>],
  [<table_constraint>], -- comment
  [<table_constraint>], /* comment */
    ⋮
  [<table_constraint>]  -- no comma
);
```

Semantics

attr1	attr2	...	attrn

<table_name>(
    <attr1> : <type1>,
    <attr2> : <type2>,
      ⋮
    <attrn> : <typen>
)

Creating/Insert/Deleting

CREATE TABLE

Example

Create the relation *Employees*(*id:* ***INT***, *name:* ***TEXT***, *age:* ***INT***, *role:* ***TEXT***).

```
CREATE TABLE Employees (
    id      INTEGER,
    name    VARCHAR(50),
    age     INTEGER,
    role    VARCHAR(50)
);
```

Table "Employees"

id	name	age	role

Creating/Insert/Deleting

Data Types

Basic Data Types

Type	Description
`BOOLEAN`	Logical Boolean (*true*/*false*)
`INTEGER` (`INT`)	Signed 4-bytes integer
`FLOAT8`	Double precision 8-bytes floating-point
`NUMERIC[(p,s)]`	Exact numeric of selectable precision
`CHAR(n)`	Fixed-length character string
`VARCHAR(n)`	Variable-length character string
`TEXT`	Variable-length character string
`DATE`	Calendar date (*year*, *month*, *day*)
`TIMESTAMP`	Date and time

Extended Data Types

Kinds	Types
Document	XML JSON
Spatial	Point Line Polygon Circle Box Path
Special	Money/Currency MAC/IP Address

User-Defined Types

```
CREATE TYPE
```

Creating/Insert/Deleting

INSERT INTO

Syntax

```
INSERT INTO <table_name> [(attr_1, attr_2, ..., attr_n)]
  VALUES
    (<val_1_1>, <val_2_1>, ..., <val_n_1>),
    (<val_1_2>, <val_2_2>, ..., <val_n_2>),
      ⋮
    (<val_1_m>, <val_2_m>, ..., <val_n_m>);
```

Notes

- Either ***all*** inserted or ***none*** inserted
- Attributes can be specified out-of-order *(optionally)*
- Missing values are replaced with `NULL` *(if allowed)* or default values *(if specified)*

Creating/Insert/Deleting

CREATE
Data Types
INSERT
- Syntax
- Semantics
- Default
DELETE

INSERT INTO

Semantics

Table "Employees"

id	name	age	role

```
INSERT INTO Employees
  VALUES (101, 'Sarah', 25, 'dev');
```

id	name	age	role
101	'Sarah'	25	'dev'

```
INSERT INTO Employees (name, id)
  VALUES ('Judy', 102), ('Max', 103);
```

id	name	age	role
101	'Sarah'	25	'dev'
102	'Judy'	NULL	NULL
103	'Max'	NULL	NULL

Creating/Insert/Deleting

INSERT INTO

Default

```
CREATE TABLE Employees (
  id    INTEGER,
  name  VARCHAR(50),
  age   INTEGER,
  role  VARCHAR(50) DEFAULT 'sales'
);
```

Table "Employees"

id	name	age	role

Insertion

```
INSERT INTO Employees
  VALUES (101, 'Sarah', 25, 'dev');
INSERT INTO Employees (name, id)
  VALUES ('Judy', 102), ('Max', 103);
```

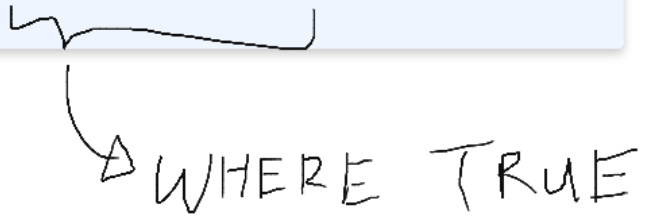
id	name	age	role
101	'Sarah'	25	'dev'
102	'Judy'	NULL	'sales'
103	'Max'	NULL	'sales'

Creating/Insert/Deleting

DELETE FROM

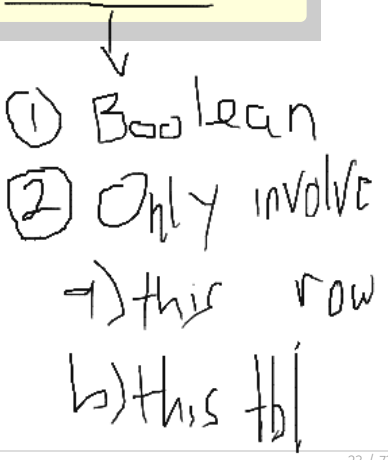
Syntax

```
DELETE FROM <table_name>
  [ WHERE <condition> ];
```



Note

- Condition is *optional*
  - If unspecified, equivalent to always true
  - If specified, condition can be *arbitrarily complex*



Creating/Insert/Deleting

CREATE
Data Types
INSERT
DELETE
- Syntax
- Semantics

DELETE FROM

Semantics

Principle of Acceptance

Perform the operation if the condition ***evaluates to True****.
- Used in `WHERE` clause *(and relational algebra)*

Table "Employees"

id	name	age	role
101	'Sarah'	25	'dev'
102	'Judy'	NULL	'sales'
103	'Max'	NULL	NULL

Example

```
DELETE FROM Employees;
-- delete all!
```

id	name	age	role

```
DELETE FROM Employees
  WHERE role <> 'dev';
```

id	name	age	role
101	'Sarah'	25	'dev'
103	'Max'	NULL	NULL

*Remember, we have three-valued logic!

Integrity Constraints

Integrity Constraints

Preliminary

Principles

Principle of Rejection

Reject the insertion if the condition ***evaluates to False***.

- Used in integrity constraints

Integrity Constraints

Preliminary

Three-Valued Logic

IS [NOT] NULL

x	x IS NULL	x IS NOT NULL
not-NULL	False	True
NULL	True	False

IS [NOT] DISTINCT FROM

x	y	x IS DISTINCT FROM y	x IS NOT DISTINCT FROM y
not-NULL	not-NULL	x <> y	x = y
not-NULL	NULL	True	False
NULL	not-NULL	True	False
NULL	NULL	False	True

Integrity Constraints

Preliminary

Types of Integrity Constraints

1. Not-NULL Constraints
2. Unique Constraints
3. Primary Key Constraints
4. Foreign Key Constraints
5. General Constraints

→ Look at another table

Specification

Where

- **Column Constraint:**
  - ~~Applies to a single column~~*
    - Specified *at* column definition
- **Table Constraint:**
  - ~~Applies to~~ one or more column
    - Specified *after* column definition

Naming

- **Named Constraint:**
  - Name is specified by *user*
- **Unnamed Constraint:**
  - Name is automatically assigned by *DBMS*

Integrity Constraints

Not-NULL Constraint

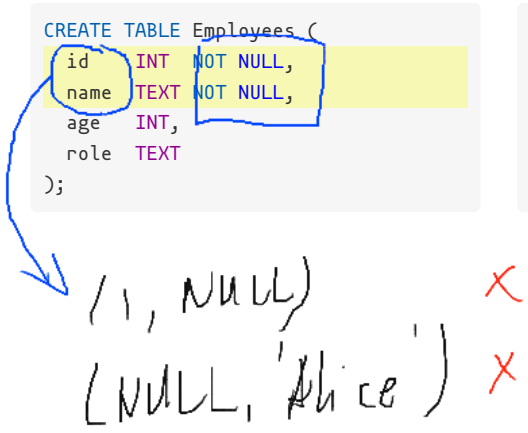
Column Constraint

Unnamed

```
CREATE TABLE Employees (
  id    INT  NOT NULL,
  name  TEXT NOT NULL,
  age   INT,
  role  TEXT
);
```

Named

```
CREATE TABLE Employees (
  id    INT  CONSTRAINT nn_id   NOT NULL,
  name  TEXT CONSTRAINT nn_name NOT NULL,
  age   INT,
  role  TEXT
);
```



Integrity Constraints

Not-NULL Constraint

Schema

Table "Employees"

```
CREATE TABLE Employees (
  id    INT  NOT NULL,
  name  TEXT NOT NULL,
  age   INT,
  role  TEXT
);
```

id	name	age	role
101	'Sarah'	25	'dev'
NULL	'Judy'	32	'sales'
102	''	27	'hr'
103	'Max'	NULL	NULL

Row
Row 1
Row 2
Row 3
Row 4

Exercise #1

Consider the schema on the left and the table at the top. Which rows violate the `NOT NULL` constraint?

	Choice	Comment	
A	Row 1		?
B	Row 2		?
C	Row 3		?
D	Row 4		?

Integrity Constraints

Unique Constraint

Column Constraint

Unnamed

```
CREATE TABLE Employees (
  id    INT  UNIQUE,
  name  TEXT,
  age   INT,
  role  TEXT
);
```

Named

```
CREATE TABLE Employees (
  id    INT  CONSTRAINT emp_id UNIQUE,
  name  TEXT ,
  age   INT,
  role  TEXT
);
```

$$\forall r_1, r_2 : r_1 \equiv r_2 \vee (\exists a : r_1.a \neq r_2.a)$$

$$\Downarrow$$

$$\text{unique}(\text{Emp})$$

Integrity Constraints

Unique Constraint

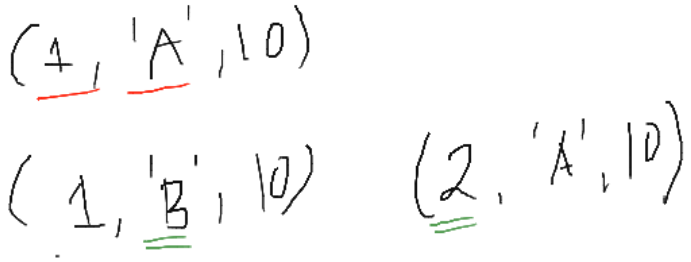
Table Constraint

Unnamed

```
CREATE TABLE Teams (
  eid   INT,
  pname TEXT,
  hours INT,
  UNIQUE (eid, pname)
);
```

Named

```
CREATE TABLE Teams (
  eid   INT,
  pname TEXT,
  hours INT,
  CONSTRAINT team_id UNIQUE (eid, pname)
);
```



Integrity Constraints

Preliminary
Not-NULL
Unique
- Column Constraint
- Table Constraint
- Exercise
Primary Key
Foreign Key
General
Summary

Constraints

Schema

Table "Teams"

```
CREATE TABLE Teams (
  eid   INT,
  pname TEXT,
  hours INT,
  UNIQUE (eid, pname)
);
```

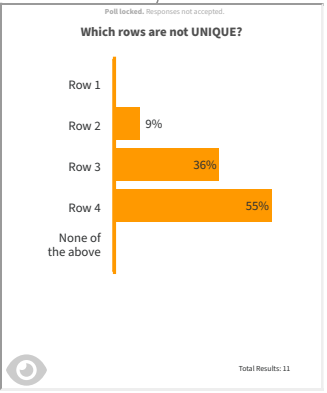
eid	pname	hours
101	NULL	25
101	NULL	25
102	'Judy'	32
102	'Judy'	32

Row
Row 1
Row 2
Row 3
Row 4

NULL <> NULL
=> NULL

Exercise #2

Consider the schema on the left and the table at the top. Which rows are not `UNIQUE`?



	Choice	Comment	
A	Row 1		?
B	Row 2		?
C	Row 3		?
D	Row 4		?

Integrity Constraints

Primary Key Constraint

Recap

Definition

A **primary key** is a *selected **candidate keys***.

- Uniquely identifies a tuple in a relation
- Cannot be `NULL`

In other words, `UNIQUE` and `NOT NULL`

Integrity Constraints

Primary Key Constraint

Column Constraint

Unnamed

```
CREATE TABLE Employees (
  id    INT  PRIMARY KEY,
  name  TEXT,
  age   INT,
  role  TEXT
);
```

Named

```
CREATE TABLE Employees (
  id    INT  CONSTRAINT emp_pk PRIMARY KEY,
  name  TEXT,
  age   INT,
  role  TEXT
);
```

Integrity Constraints

Primary Key Constraint

Table Constraint

Unnamed

```
CREATE TABLE Teams (
  eid   INT,
  pname TEXT,
  hours INT,
  PRIMARY KEY (eid, pname)
);
```

Named

```
CREATE TABLE Teams (
  eid   INT,
  pname TEXT,
  hours INT,
  CONSTRAINT team_pk PRIMARY KEY (eid, pname)
);
```

Integrity Constraints

Primary Key Constraint

Equivalent Constraint

Unnamed

```
CREATE TABLE Teams (
  eid   INT  NOT NULL,
  pname TEXT NOT NULL,
  hours INT,
  UNIQUE (eid, pname)
);
```

Named

```
CREATE TABLE Teams (
  eid   INT  CONSTRAINT nn_eid   NOT NULL,
  pname TEXT CONSTRAINT nn_pname NOT NULL,
  hours INT,
  CONSTRAINT team_pk PRIMARY KEY (eid, pname)
);
```

Integrity Constraints

Foreign Key Constraint

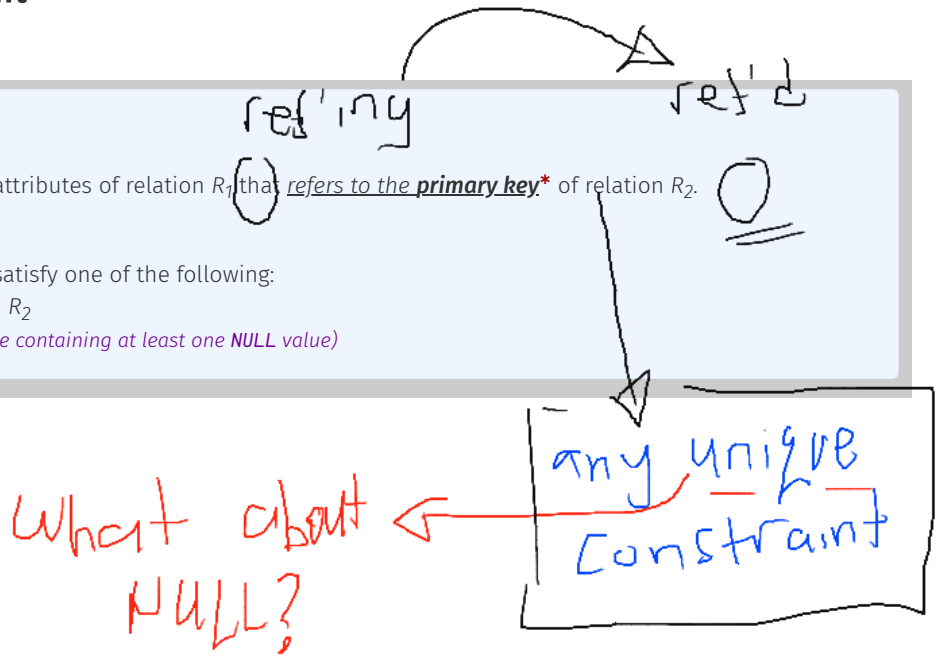
Recap

Definition

A **foreign key** is a *subset* of attributes of relation R_1 that *refers to the* ***primary key**** of relation R_2.

Requirement:
Each foreign key in R_1 must satisfy one of the following:

- Appear as ***primary key*** in R_2
- Be a `NULL` value *(or a tuple containing at least one `NULL` value)*

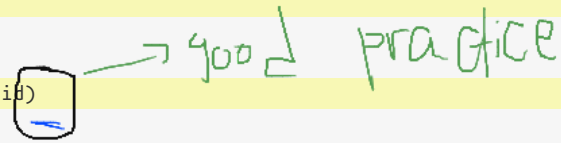


Integrity Constraints

Foreign Key Constraint

Constraint on Referencing Relation

```
CREATE TABLE Teams (
  eid     INT,
  pname   TEXT REFERENCES Projects,
  hours   INT,
  PRIMARY KEY (eid, pname),
  FOREIGN KEY (eid) REFERENCES Employees (id)
);
```



Referenced Relations

```
CREATE TABLE Employees (
  id    INT  PRIMARY KEY,
  name  TEXT,
  age   INT,
  role  TEXT
);
```

```
CREATE TABLE Projects (
  name        TEXT,
  start_year  INT,
  end_year    INT,
  PRIMARY KEY (name)
);
```

Integrity Constraints

Foreign Key Constraint

Condition

A referential constraint is satisfied if one of the following conditions is true, depending on the <match option> specified in the <referential constraint definition>:

- If no <match type> was specified then, for each row R1 of the referencing table, either at least one of the values of the referencing columns in R1 shall be a null value, or the value of each referencing column in R1 shall be equal to the value of the corresponding referenced column in some row of the referenced table.

- If MATCH FULL was specified then, for each row R1 of the referencing table, either the value of every referencing column in R1 shall be a null value, or the value of every referencing column in R1 shall not be null and there shall be some row R2 of the referenced table such that the value of each referencing column in R1 is equal to the value of the corresponding referenced column in R2.

Integrity Constraints

Foreign Key Constraint

Problems

- What if the first tuple on "Movies" is **deleted**?
- What if Sigourney Weaver *id* is **updated** from 20 to 30?

Idea

> Extend the syntax to specify the *behavior* when data in referenced table is deleted or updated using *optional* specification:
>
> - `ON DELETE <action>`
> - `ON UPDATE <action>`

Table "Movies"

id	title	genre	opened
101	Aliens	Action	1986
102	Logan	Drama	2017
103	Heat	Crime	1995
104	Terminator	Action	1984

Table "Cast"

movie_id	actor_id	role
101	20	Ellen Ripley
101	23	Private Hudson
101	54	Corporal Hicks
102	21	Logan
104	23	Punk Leader

Table "Actors"

id	name	dob
20	Sigourney Weaver	08-10-1949
21	Hugh Jackman	12-10-1968
22	Tom Hanks	09-07-1956
23	Bill Paxton	17-05-1955

Integrity Constraints

Foreign Key Constraint

Actions

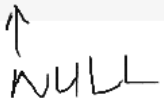
Keyword	Action
`NO ACTION`	*Reject* delete/update if it violates constraint *(default value)*
`RESTRICT`	Similar to "`NO ACTION`" except that check of constraint *cannot be deferred* *(deferrable constraints are discussed in a bit)*
`CASCADE`	*Propagates* delete/update to the referencing tuples
`SET DEFAULT`	*Updates* the foreign key of the referencing tuples to some default value *(**Important:** default value must be a primary key in the referencing table)*
`SET NULL`	*Updates* the foreign key of the referencing tuples to `NULL` value *(**Important:** corresponding column must be allowed to contain **NULL** values)*

Integrity Constraints

Foreign Key Constraint

Example

```
CREATE TABLE Teams (
  eid    INT,
  pname  TEXT DEFAULT 'FastCash',
  hours  INT,
  PRIMARY KEY (eid, pname),
  FOREIGN KEY (eid) REFERENCES Employees (id) ON DELETE NO ACTION ON UPDATE CASCADE,
  FOREIGN KEY (pname) REFERENCES Projects (name) ON DELETE SET DEFAULT ON UPDATE CASCADE
);
```



Effects

- Updates on *Employees.id* and *Projects.name* are *propagated*
- Deleting a project will set *Teams.pname* to *default value* *(i.e., FastCash)*
- Deleting an employee will *raise an error* if that employee is still assigned to a team

Integrity Constraints

Foreign Key Constraint

Example

```
CREATE TABLE Teams (
  eid    INT,
  pname  TEXT DEFAULT 'FastCash',
  hours  INT,
  PRIMARY KEY (eid, pname),
  FOREIGN KEY (eid) REFERENCES Employees (id) ON DELETE NO ACTION ON UPDATE CASCADE,
  FOREIGN KEY (pname) REFERENCES Projects (name) ON DELETE SET DEFAULT ON UPDATE CASCADE
);
```

name	start_year	end_year
BigAI	2020	2025
FastCash	2018	2025
⋮	⋮	⋮

eid	pname	hours
101	BigAi	10
102	GlobalDB	20
⋮	⋮	⋮

```
UPDATE Projects
SET    name = 'SmartAI'
WHERE  name = 'BigAI';
```

name	start_year	end_year
SmartAI	2020	2025
FastCash	2018	2025
⋮	⋮	⋮

eid	pname	hours
101	SmartAI	10
102	GlobalDB	20
⋮	⋮	⋮

Integrity Constraints

Foreign Key Constraint

Example

```
CREATE TABLE Teams (
  eid    INT,
  pname  TEXT DEFAULT 'FastCash',
  hours  INT,
  PRIMARY KEY (eid, pname),
  FOREIGN KEY (eid) REFERENCES Employees (id) ON DELETE NO ACTION ON UPDATE CASCADE,
  FOREIGN KEY (pname) REFERENCES Projects (name) ON DELETE SET DEFAULT ON UPDATE CASCADE
);
```

name	start_year	end_year
BigAI	2020	2025
FastCash	2018	2025
⋮	⋮	⋮

eid	pname	hours
101	BigAi	10
102	GlobalDB	20
⋮	⋮	⋮

```
DELETE FROM Projects
WHERE  name = 'BigAI';
```

name	start_year	end_year
FastCash	2018	2025
⋮	⋮	⋮

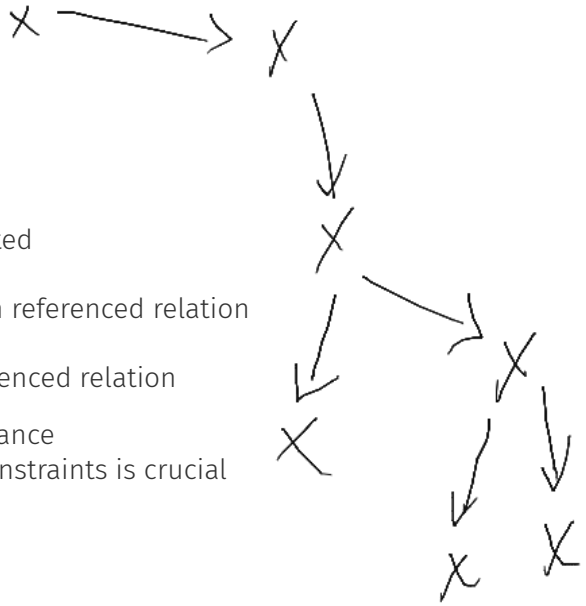
eid	pname	hours
101	**FastCash**	10
102	GlobalDB	20
⋮	⋮	⋮

Integrity Constraints

Foreign Key Constraint

Practical Considerations

- Specified constraints might not behave as expected
  - `SET NULL` issue with prime attributes
  - `SET DEFAULT` issue with default value not in referenced relation
  - `CASCADE` may create a *chain* of propagation
    - If the referencing relation is also a referenced relation
- `CASCADE` may *significantly* affect overall performance
- Careful defign and specification of foreign key constraints is crucial



Integrity Constraints

Foreign Key Constraint

Schema

- *R(A, B)*
- *S(C, D)*

A	B
1	2
2	1
1	1
1	3

C	D
3	NULL
NULL	1
1	3
NULL	NULL

Row
Row 1
Row 2
Row 3
Row 4

Exercise #3

Consider the foreign key constraints *(S.C, S.D)* ⇝ *(R.A, R.B)*. Which rows violate the `FOREIGN KEY` constraint?

	Choice	Comment	
A	Row 1		?
B	Row 2		?
C	Row 3		?
D	Row 4		?

Integrity Constraints

General Constraint

CHECK Constraint

- Most basic general constraint *(i.e., not a structural integrity constraint)*
- Allows us to specify that column values must satisfy *an arbitrary Boolean expression*
- **Scope:** *one table, one row*

Single Column

```
CREATE TABLE Projects (
  eid    INT,
  pname  TEXT,
  hours  INT CHECK (hours > 0),
  PRIMARY KEY (eid, pname)
);
```

Multiple Columns

```
CREATE TABLE Teams (
  name        TEXT PRIMARY KEY,
  start_year  INT,
  end_year    INT,
  CHECK (start_year <= end_year)
);
```

Integrity Constraints

General Constraint

Complex Boolean Constraint

> **Example**
>
> Create the Table "Teams" where the minimum hours for 'CoreOS' is at least 30 hours but there are no such restrictions for other projects.

```
CREATE TABLE Projects (
  eid    INT,
  pname  TEXT,
  hours  INT CHECK (hours > 0),
  PRIMARY KEY (eid, pname),
  CHECK (
    (pname = 'CoreOS' AND hours >= 30)
    OR
    (pname <> 'CoreOS' AND hours > 0)
  )
);
```

[#] Is this good enough? Can you do better?

Integrity Constraints

General Constraint

Assertion

- `CREATE ASSERTION` statement *(since SQL-92)*
- Used for *(almost)* arbitrary constraints *(multiple table, multiple rows)*
  - **Example:** *"Each project must have at least one team member being 30 or older"*

In Practice

- Assertions may have various potential side-effects and/or limitations
  - Cannot modify the data
  - No proper error handling
  - Not linked to a specific table *(e.g., dropping a table does not affect assertion)*
- Most RDBMS do **NOT** support assertions
  - **Alternative: Triggers**

Integrity Constraints

Summary

Types

Type	Column	Table	Condition
Not-NULL	`NOT NULL`	-	`IS NOT NULL`
Unique	`UNIQUE`	`UNIQUE(`A_1`,`A_2`,...)`	`x.`A_i `<> y.`A_i
Primary Key	`PRIMARY KEY`	`PRIMARY KEY(`A_1`,`A_2`,...)`	`UNIQUE` & `NOT NULL`
Foreign Key	`REFERENCES` R_1`(B)`	`FOREIGN KEY (`A_1`,`A_2`,...)` `REFERENCES` R_1`(`B_1`,`B_2`,...)`	The tuple exists in R_1 or the tuple contains `NULL` value
General	`CHECK (c)`	`CHECK (c)`	Condition `c` does not evaluate to `False`

Modifying Database

Modifying Database

ALTER TABLE

Syntax

```
ALTER TABLE <table_name>
  [ALTER / ADD / DROP] [COLUMN / CONSTRAINT] <name>
  <changes>;
```

Common Changes:

- Adding/dropping columns
- Adding/dropping constraints
- Changing the specification of a single column:
  - Data type
  - Default values

Modifying Database

ALTER TABLE

Altering a Single Column

Changing Data Type

```
ALTER TABLE Projects ALTER COLUMN name TYPE VARCHAR(200);
```

Changing Default Values

```
ALTER TABLE Projects ALTER COLUMN start_year SET DEFAULT 2021;
```

Removing Default Values

```
ALTER TABLE Projects ALTER COLUMN name DROP DEFAULT;
```

Modifying Database

ALTER TABLE

Add/Drop

Adding/Dropping Columns

```
ALTER TABLE Projects ADD COLUMN budget NUMERIC DEFAULT 0.0;
ALTER TABLE Projects DROP COLUMN budget;
```

Adding/Dropping Constraints

```
ALTER TABLE Teams ADD CONSTRAINT
  eid_fkey FOREIGN KEY (eid) REFERENCES Employees (id);
ALTER TABLE Teams DROP CONSTRAINT eid_fkey;
```

- Name of constraints need to be known *(may be retrieved from metadata)*

[#] https://www.postgresql.org/docs/14/sql-altertable.html

Modifying Database

ALTER
DROP
- Syntax

DROP TABLE

Syntax

```
DROP TABLE
  [IF EXISTS]   -- no error if table does not exist
  <table_name>[, <table_name> [, <table_name> [...]]] -- multiple table
  [CASCADE];    -- also delete referencing table
```

Deferrable Constraints

Deferrable Constraints

Motivation

Definition

An SQL **statement** is any SQL code that ends with a ***semi-colon*** *(i.e.,* `;`*)*

Default Behavior

- Constraints are checked **immediately** at the end of SQL statement
  - This is true even for transaction containing multiple SQL statement A violation will cause the statement/transaction to be **rolled back**

Relaxed Constraint

- Check only at the end of a transaction
- Available for: `UNIQUE`, `PRIMARY KEY`, `FOREIGN KEY`

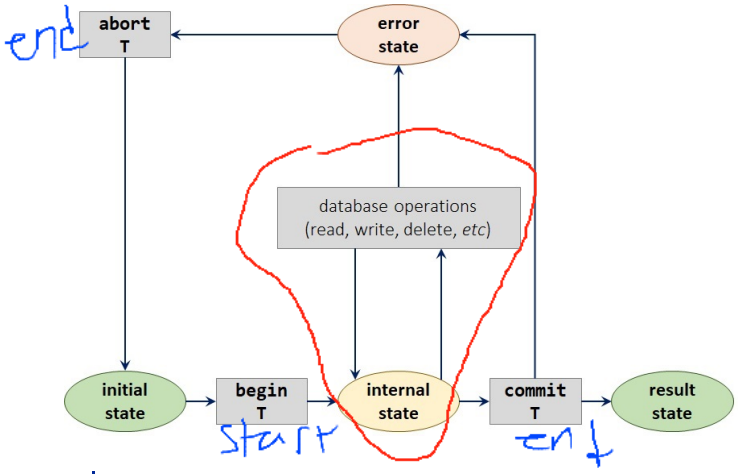
Deferrable Constraints

Motivation
- *Behavior*
- ***Transition***
- *Circular*

Deferment Considerations

Motivation

Transition Graph



> Deferrable constraints may *(temporarily)* be violated within the scope of a transaction in the "internal state"

Deferrable Constraints

Motivation

A Curious Case of Circular Reference

Employee

> An employee must work in a department

Department

> A department must be managed by an employee

```
CREATE TABLE Employee (
  eid   INT PRIMARY KEY,
  name  TEXT,
  did   INT
);
```

```
CREATE TABLE Department (
  did   INT PRIMARY KEY,
  name  TEXT,
  eid   INT
);
```

```
ALTER TABLE Employee ADD CONSTRAINT
  did_fk FOREIGN KEY (did)
  REFERENCES Department (did);
```

```
ALTER TABLE Department ADD CONSTRAINT
  eid_fk FOREIGN KEY (eid)
  REFERENCES Employee (eid);
```

Deferrable Constraints

Deferment

NOT DEFERRABLE

```
BEGIN;  -- start of transaction

INSERT INTO Employee VALUES (101, 'Sarah', 1001);
INSERT INTO Department VALUES (1001, 'dev', 101);
COMMIT; -- successful end of transaction
```

- **Line 3:** Foreign Key Constraint Violation
- **Line 4:** Foreign Key Constraint Established

ABORT
not executed

```
ALTER TABLE Employee ADD CONSTRAINT
  did_fk FOREIGN KEY (did)
  REFERENCES Department (did)
  NOT DEFERRABLE;
```

```
ALTER TABLE Department ADD CONSTRAINT
  eid_fk FOREIGN KEY (eid)
  REFERENCES Employee (eid)
  NOT DEFERRABLE;
```

Deferrable Constraints

Deferment

INITIALLY DEFERRED

```
BEGIN;  -- start of transaction

INSERT INTO Employee VALUES (101, 'Sarah', 1001);
INSERT INTO Department VALUES (1001, 'dev', 101);
COMMIT; -- successful end of transaction
```

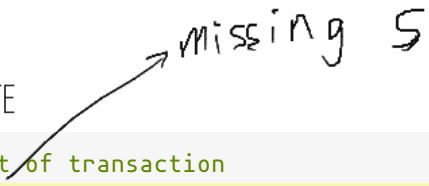
- **Line 3:** Foreign Key Constraint Violation — *constraint check deferred*
- **Line 4:** Foreign Key Constraint Established — **SUCCESS**

```
ALTER TABLE Employee ADD CONSTRAINT
  did_fk FOREIGN KEY (did)
  REFERENCES Department (did)
  DEFERRABLE INITIALLY DEFERRED;
```

```
ALTER TABLE Department ADD CONSTRAINT
  eid_fk FOREIGN KEY (eid)
  REFERENCES Employee (eid)
  DEFERRABLE INITIALLY DEFERRED;
```

Deferrable Constraints

Deferment



INITIALLY IMMEDIATE

```
1 BEGIN;  -- start of transaction
2 SET CONSTRAINT eid_fk DEFERRED;
3 INSERT INTO Employee VALUES (101, 'Sarah', 1001);
4 INSERT INTO Department VALUES (1001, 'dev', 101);
5 COMMIT; -- successful end of transaction
```

- **Line 3:** Foreign Key Constraint Violation — *constraint check deferred*
- **Line 4:** Foreign Key Constraint Established — **SUCCESS**

```
ALTER TABLE Employee ADD CONSTRAINT
  did_fk FOREIGN KEY (did)
  REFERENCES Department (did)
  DEFERRABLE INITIALLY IMMEDIATE;
```

```
ALTER TABLE Department ADD CONSTRAINT
  eid_fk FOREIGN KEY (eid)
  REFERENCES Employee (eid)
  DEFERRABLE INITIALLY DEFERRED;
```

Deferrable Constraints

Considerations

Benefits

- No need to care about the order of SQL statements within a transaction
  (i.e., intermediate state may temporarily violate constraints)
- Allows for cyclic foreign key constraints
- Performance boost when constraint checks are bottleneck
  (e.g., batch insert of large number of tuples)

Potential Downsides

- Troubleshooting can be more difficult
- Data definition no longer unambiguous
- May incur performance penalty when performing queries
  - Certain checks may need to be done at run-time especially during the time when the constraint check is deferred

Summary

Summary

SQL

What Is It?

- The standard language for RDBMS
  - Different language groups: DDL, DML, DQL, DCL, TCL

- **DDL:** `CREATE TABLE`, `ALTER TABLE`, `DROP TABLE`
- **DML:** `INSERT`, `UPDATE`, `DELETE`

Key Challenge

- Specification of integrity constraints
  - `NOT NULL`, `UNIQUE`, `PRIMARY KEY`, `FOREIGN KEY`, `CHECK`

- Specification actions in case of foreign key constraint violations *(`ON UPDATE`/`ON DELETE`)*
- Relaxed checks of violations with deferrable constraints

Summary

SQL

Table Syntax

CREATE

```
CREATE TABLE <table_name> (
  <attr> <type> [<column_constraint>],
  <attr> <type> [<column_constraint>],
    :
  [<table_constraint>],
  [<table_constraint>],
    :
);
```

DROP

```
DROP TABLE [IF EXISTS] <table_name>;
```

Summary

SQL

Rows Syntax

INSERT

```
INSERT INTO <table_name> [(<attr>, ...)]
  VALUES (<values>, ...) [, (<values>, ...)];
```

UPDATE

```
UPDATE <table_name>
  SET <attr> = <value> [, <attr> = <value>]
  [WHERE <condition>];
```

DELETE

```
DELETE FROM <table_name> [WHERE <condition>];
```

Summary

SQL
Constraints

Integrity Constraints

Types

Type	Column	Table	Condition
Not-NULL	NOT NULL	-	IS NOT NULL
Unique	UNIQUE	UNIQUE(A_1,A_2,...)	x.A_i <> y.A_i
Primary Key	PRIMARY KEY	PRIMARY KEY(A_1,A_2,...)	UNIQUE & NOT NULL
Foreign Key	REFERENCES R_1(B)	FOREIGN KEY (A_1,A_2,...) REFERENCES R_1(B_1,B_2,...)	The tuple exists in R_1 or the tuple contains NULL value
General	CHECK (c)	CHECK (c)	Condition c does not evaluate to False

```
postgres=# exit
Press any key to continue . . .
```

Solutions

Not-NULL Constraint

Schema

Table "Employees"

```
CREATE TABLE Employees (
  id    INT  NOT NULL,
  name  TEXT NOT NULL,
  age   INT,
  role  TEXT
);
```

id	name	age	role
101	'Sarah'	25	'dev'
NULL	'Judy'	32	'sales'
102	''	27	'hr'
103	'Max'	NULL	NULL

Row
Row 1
Row 2
Row 3
Row 4

> **Exercise #1**
>
> Consider the schema on the left and the table at the top. Which rows violate the `NOT NULL` constraint?

	Choice	Comment	
A	Row 1	*NO: there is no **NULL** values*	✗
B	Row 2	*YES: `id` is **NULL***	✓
C	Row 3	*NO: empty string is not **NULL***	✗
D	Row 4	*NO: these attributes can be **NULL***	✗

Solutions

Exercise #1
Exercise #2
Exercise #3

Constraints

Schema

Table "Teams"

```
CREATE TABLE Teams (
  eid   INT,
  pname TEXT,
  hours INT,
  UNIQUE (eid, pname)
);
```

eid	pname	hours
101	NULL	25
101	NULL	25
102	'Judy'	32
102	'Judy'	32

Row
Row 1
Row 2
Row 3
Row 4

> **Exercise #2**
>
> Consider the schema on the left and the table at the top. Which rows are not `UNIQUE`?

	Choice	Comment	
A	Row 1	*NO: NULL <> NULL evaluates to NULL*	✗
B	Row 2	*NO: NULL <> NULL evaluates to NULL*	✗
C	Row 3	*YES: equal to Row 4*	✓
D	Row 4	*YES: equal to Row 3*	✓

Solutions

Exercise #1
Exercise #2
Exercise #3

Foreign Key Constraint

A	B
1	2
2	1
1	1
1	3

C	D
3	NULL
NULL	1
1	3
NULL	NULL

Row
Row 1
Row 2
Row 3
Row 4

Schema

- *R(A, B)*
- *S(C, D)*

Exercise #3

Consider the foreign key constraints *(S.C, S.D)* ⇝ *(R.A, R.B)*. Which rows violate the `FOREIGN KEY` constraint?

	Choice	Comment	
A	Row 1	*NO: it has **NULL** value for column D*	✗
B	Row 2	*NO: it has **NULL** value for column D*	✗
C	Row 3	*NO: the tuple (1, 3) appears in R*	✗
D	Row 4	*NO: it is both **NULL***	✗