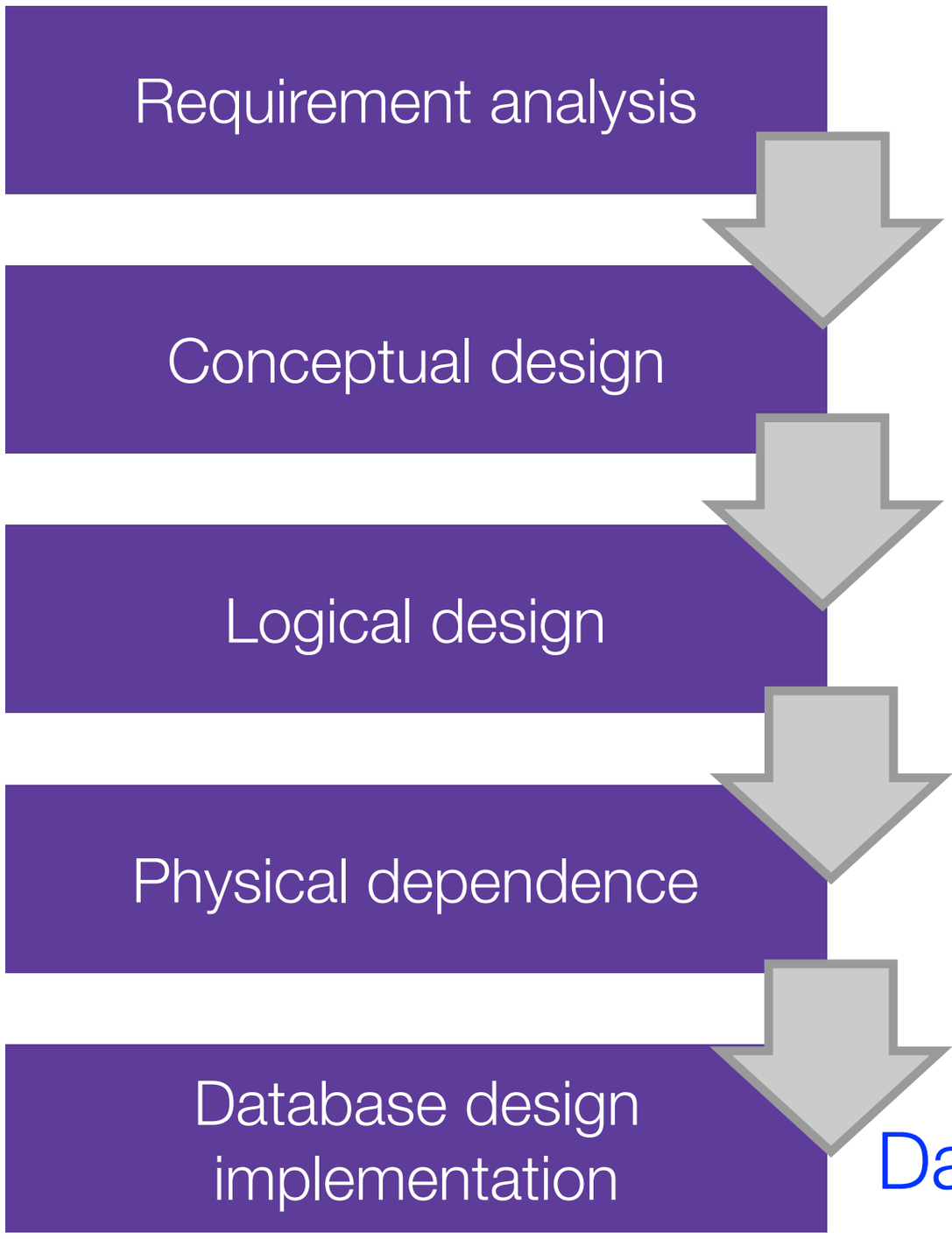
SQL Introduction

CS 377: Database Systems

Recap: Last Two Weeks

Recap: Last Two Weeks



Requirement specification

Conceptual data model
(**ER Model**)

Representation data model
(**Relational Model**)

Physical data model

Data definition / Manipulation (SQL)

Today's Lecture

1. SQL
    1. Introduction
    2. Database Creation
    3. Example: Company Database

SQL Introduction

SQL

- SQL stands for Structured Query Language
- Standard language for querying and manipulating data
- Most widely used database language and is the de facto standard
- Very high-level programming language
- Based on relational algebra (or relational calculus)

SQL: Not Just for Querying

- Data definition language (define conceptual model of database)

- Data manipulation language (insert, update, delete data into conceptual model of database)

- View definition language (define views or external schemas to support logical data independence)

SQL History

- One of the first commercial languages for Codd's relational model
- Originally developed by IBM
- Many SQL standards: SQL-92, SQL:1999, SQL:2011
  - Vendors support different subsets

SQL Usage

- Stand-alone: user enters SQL commands via a command line or in a GUI

- Embedded in a host language: SQL commands are embedded (written inside) an “ordinary” program in a high level language (e.g., Java, C++, C, etc.)

- Library-based: SQL commands are made available through library functions (e.g., Java, Python)

- Web-based: various languages with extensions allow webpages to access database server

SQL vs Relational Model

- SQL relation (table) is a multi-set (bag) of tuples; it is not a set of tuples (i.e., tuples may appear more than once)
    - Bags (rather than sets, which are easier to handle) is favored because of database efficiency
    - Duplicate elimination is costly (requires time and memory), so it is only best to be used when necessary
- SQL relations can be constrained to sets by specifying PRIMARY KEY or UNIQUE attributes, or using the DISTINCT option in a query

SQL DBMS

- MySQL is the most popular, freely available database management system
  - Common choice for many web applications and well-known websites including Google, Facebook, Wikipedia, and YouTube
- SQLite is a very powerful, embedded relational database management system which is fast and efficient but does not support user management
- PostgreSQL is the most advanced, SQL-compliant and open-source objective RDBMS with complete support for reliable transactions but not as efficient as MySQL

https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems

Data Definition

- Create a database
- Create new relations (tables) in a database
- Define conditions on attributes in the relations
- Alter the structure of (existing) relations
- Delete relations

CREATE SCHEMA: Creating a Database

- A database schema is used to group together database tables
    - A database schema also contains other constructs (such as indices)
        - Example: The Company database schema (see relational model slides)
- Syntax: **CREATE SCHEMA schema_name AUTHORIZATION db_user;**
- Typically executed by DBA who will grant authorities to database user who then owns schema

MySQL: CREATE SCHEMA

- MySQL version of create schema
  CREATE DATABASE database_name;

- Database is created by the root user

- Authorization is granted separately using the grant command
  GRANT permission ON database.table TO ‘user’@‘host’;

CREATE TABLE: Create a Relation

- Create a new relation by giving it a name and specifying each of its attributes and their data types
  - Relation created will be initially empty
- Syntax:

CREATE TABLE relation_name
(
attr_name$_1$ type$_1$ [attr_constraint$_1$],
attr_name$_2$ type$_2$ [attr_constraint$_2$],
…
attr_name$_n$ type$_n$ [attr_constraint$_n$],
);

Data Types in SQL: Numeric Types

- **TINYINT** (1 byte), **SMALLINT** (2 bytes), **MEDIUMINT** (3 bytes), **INTEGER** or **INT** (4 bytes), BIGINT (8 bytes) are different representations of integers

- **DECIMAL**(i,j) or **DEC**(i,j) or **NUMERIC**(i,j) are fixed point numbers with i decimal digits precision (accurate and do not have round off errors)

- **FLOAT** (8 byte) or **REAL** (4 byte) are single precision floating point numbers with roundoff errors

- **DOUBLE PRECISION** are double precision floating point numbers with roundoff errors

Data Types in SQL: Strings

- Character Strings
  - **CHARACTER(n)** or **CHAR(n)** are fixed length character strings
  - **VARCHAR(n)** or **CHAR VARYING(n)** or **CHARACTER VARYING(n)** are variable length character strings with maximum number of characters in string = n
- Bit String
  - **BIT(n)** is fixed length bit string
  - **BIT VARYING(n)** is variable length bit string

Data Types in SQL: Boolean & Date

- **BOOLEAN** is boolean data attribute
  - Due to NULL value, SQL uses three value logic to evaluate boolean expressions. If either x or y is NULL, some logical comparisons evaluate to UNKNOWN
- **DATE** is a calendar date and should be specified as ‘YYYY-MM-DD’
- **TIME** is the time of the day and specified as ‘HH:MM:SS’
- **TIMESTAMP** is DATE + TIME and specified as ‘YYYY-MM-DD HH:MM:SS’

Specifying Constraints

- Attribute constraints
  - Not null
  - Attribute domain
  - Default values
- Key attributes
- Referential integrity constraint (foreign keys)

Attribute Constraints

- **NOT NULL**: attribute cannot be assigned a NULL value
  Example: **CREATE TABLE text**
  (ssn CHAR(9) NOT NULL, …);

- **DEFAULT**: specify a default value of an attribute
  Example: **CREATE TABLE text**
  (ssn CHAR(9) NOT NULL,
  salary DECIMAL(6,2) DEFAULT 50000, …);

- **CHECK**: check if the value of an attribute is within specified range
  Example: **CREATE TABLE text**
  (ssn CHAR(9) NOT NULL,
  dno INTEGER CHECK (dno > 0 and dno < 10), …);

Key Constraints

- **PRIMARY** attribute specifies the primary key constraint
  - Syntax:
    CONSTRAINT [constraint_name] PRIMARY KEY(attribute-list)
- **UNIQUE** constraint can be used to specify candidate keys
  - Syntax:
    CONSTRAINT [constraint_name] UNIQUE(attribute-list)

Example: Key Constraint

```
CREATE TABLE test1
( ssn CHAR(9),
  salary DECIMAL(10,2),
  CONSTRAINT test1PK PRIMARY KEY(ssn));
```

```
CREATE TABLE test2
( pno INTEGER,
  pname CHAR(20),
  CONSTRAINT test2PK PRIMARY KEY(pno),
  CONSTRAINT test2PK UNIQUE(pname));
```

Referential Constraint

- **FOREIGN KEY** is used to identify tuples in another relation and such that the referenced tuples must exist to maintain integrity

- Each key constraint may be (and probably should be) identified by a constraint name

- Syntax:
  CONSTRAINT [constraint_name] FOREIGN KEY (attribute-list) REFERENCES relation(attribute-list)

Example: Referential Constraint

```
CREATE TABLE test1
( ssn CHAR(9),
  salary DECIMAL(10,2),
  CONSTRAINT test1PK PRIMARY KEY(ssn));
```

```
CREATE TABLE test3
( essn CHAR(9),
  pno INTEGER,
  CONSTRAINT test3FK
        FOREIGN KEY(essn)
        REFERENCES test1(ssn));
```

ALTER TABLE: Modify Existing Relations

- Add attributes
- Remove attributes
- Add constraints
- Remove constraints

You can not rename or update attributes in SQL!

ALTER TABLE: Add Attributes

- Used to add an attribute to one of the base relations

- New attributes will have NULLs in the tuples of the relation right after the command is executed —> NOT NULL constraint is not allowed for such an attribute

- Syntax:
  ALTER TABLE relation_name ADD attribute_name type

- Example:
  ALTER TABLE EMPLOYEE ADD JOB VARCHAR(12);

ALTER TABLE: Remove Attribute

- Syntax:
  ALTER TABLE table_name DROP [COLUMN] attr_name {RESTRICTED | CASCADE};

- **RESTRICTED**: only the attribute table_name.attr_name is dropped. However, if the attribute is part of a foreign key of another relation, it cannot be dropped

- **CASCADE**: the attribute table_name.attr_name is dropped and if the attribute table_name.attr_name is part of a foreign key in some other relation, that attribute will also be dropped.

ALTER TABLE: Add/Remove Constraints

- Add a constraint to a table: if the constraint is violated by some existing tuple in the relation, the new constraint is NOT recorded
  - Syntax:
    ALTER TABLE table_name ADD CONSTRAINT constraint_name constraint_def;
- Removing an existing constraint: this can only be done if you have given it a name at the time of definition
  - Syntax:
    ALTER TABLE table_name DROP CONSTRAINT constraint_name;

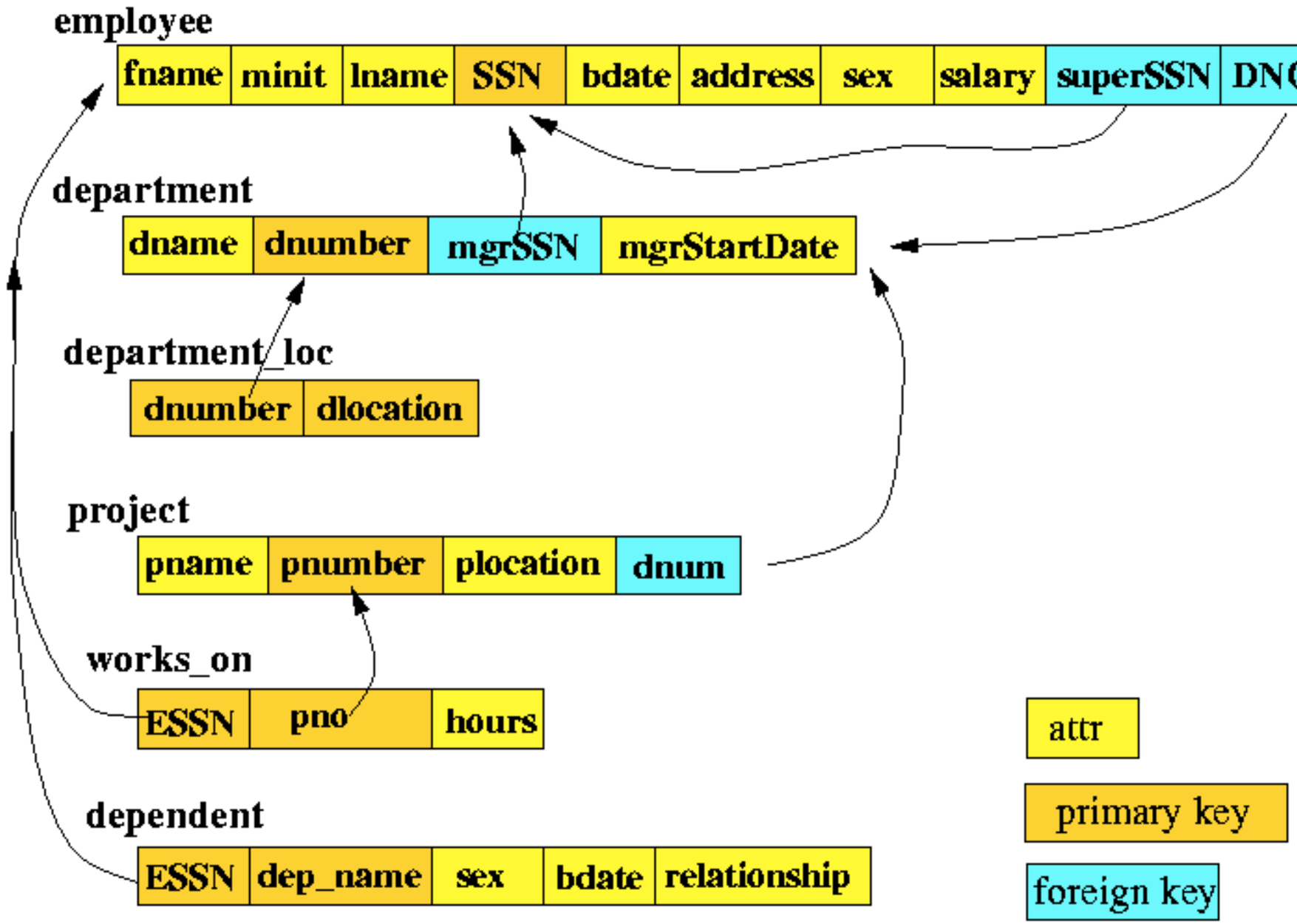
DROP TABLE: Remove a Relation

- Used to remove a relation, all its contents, and its definition

- Relation can no longer be used in queries, updates, or any other commands since its description no longer exists

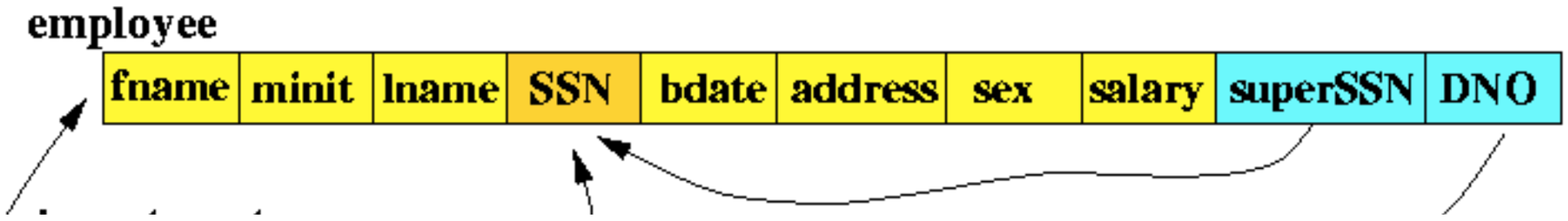
- Syntax:
  DROP TABLE table_name;
  DROP TABLE table_name cascade constraints;

Example: Company Database Schema



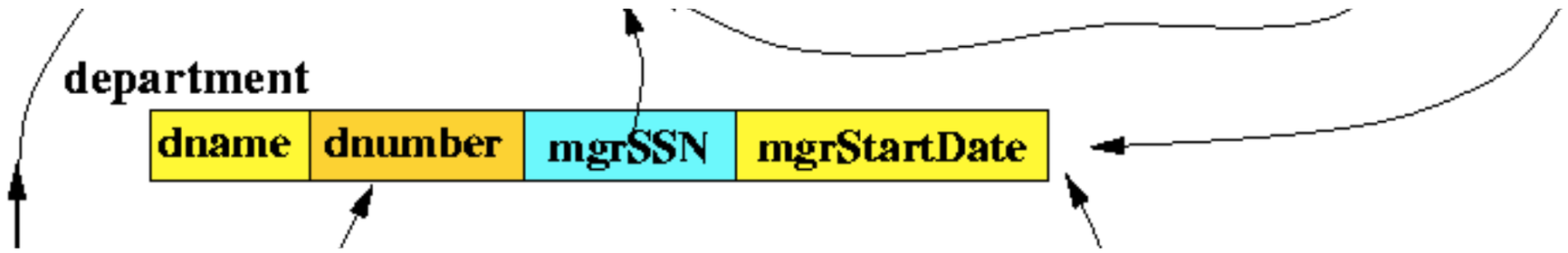
http://www.mathcs.emory.edu/~cheung/Courses/377/Syllabus/3-Relation/rel-db-design2.html

Example: Company Database (1)




```
CREATE TABLE Employee
(  Fname          VARCHAR(15)        NOT NULL,
   Minit          CHAR,
   Lname          VARCHAR(15)        NOT NULL,
   Ssn            CHAR(9)            NOT NULL,
   Bdate          DATE,
   Address        VARCHAR(30),
   Sex            CHAR,
   Salary         DECIMAL(10,2),
   Super_ssn      CHAR(9),
   Dno            INT                NOT NULL,
CONSTRAINT EmpPK PRIMARY KEY (Ssn),
CONSTRAINT EmpSuperFK FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(SSN),
CONSTRAINT EmpDeptFK FOREIGN KEY (Dno) REFERENCES DEPARTMENT(Dnumber));
```

Example: Company Database (2)

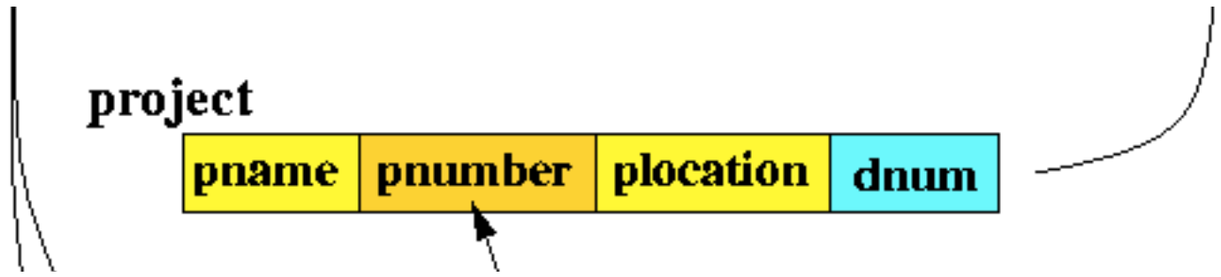



```
CREATE TABLE Department
(  Dname              VARCHAR(15)        NOT NULL,
   Dnumber            INT                NOT NULL,
   Mgr_ssn            CHAR(9)            NOT NULL,
   Mgr_start_date     DATE,
  CONSTRAINT DeptPK PRIMARY KEY (Dnumber),
  CONSTRAINT DeptNameSK UNIQUE(Dname),
  CONSTRAINT DeptMgrFK FOREIGN KEY (Mgr_ssn)
      REFERENCES EMPLOYEE(Ssn));
```

Example: Company Database (3)

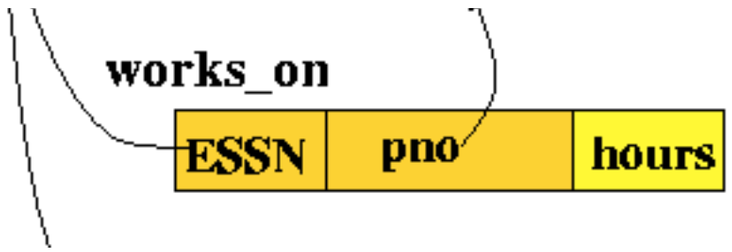

```
CREATE TABLE Dept_Locations
(  Dnumber          INT                 NOT NULL,
   Dlocation        VARCHAR(15)         NOT NULL,
  CONSTRAINT DeptLocPK
     PRIMARY KEY (Dnumber, Dlocation),
  CONSTRAINT DeptLocFK FOREIGN KEY (Dnumber)
      REFERENCES Department(Dnumber));
```

Example: Company Database (4)



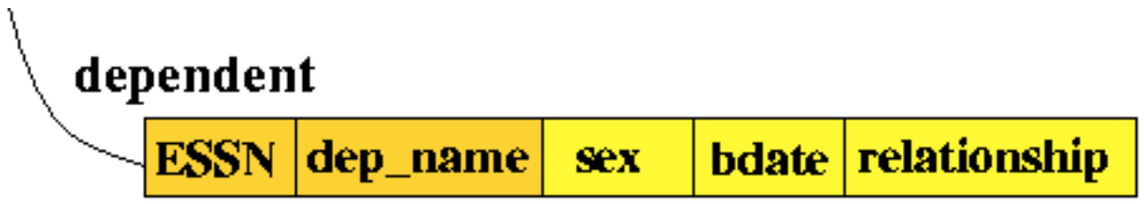

```
CREATE TABLE Project
(  Pname           VARCHAR(15)          NOT NULL,
   Pnumber         INT                  NOT NULL,
   Plocation       VARCHAR(15),
   Dnum            INT,
  CONSTRAINT ProjectPK PRIMARY KEY (Pnumber),
  CONSTRAINT ProjectSK UNIQUE(Pname),
  CONSTRAINT ProjDeptFK FOREIGN KEY (Dnum)
      REFERENCES Department(Dnumber));
```

Example: Company Database (5)



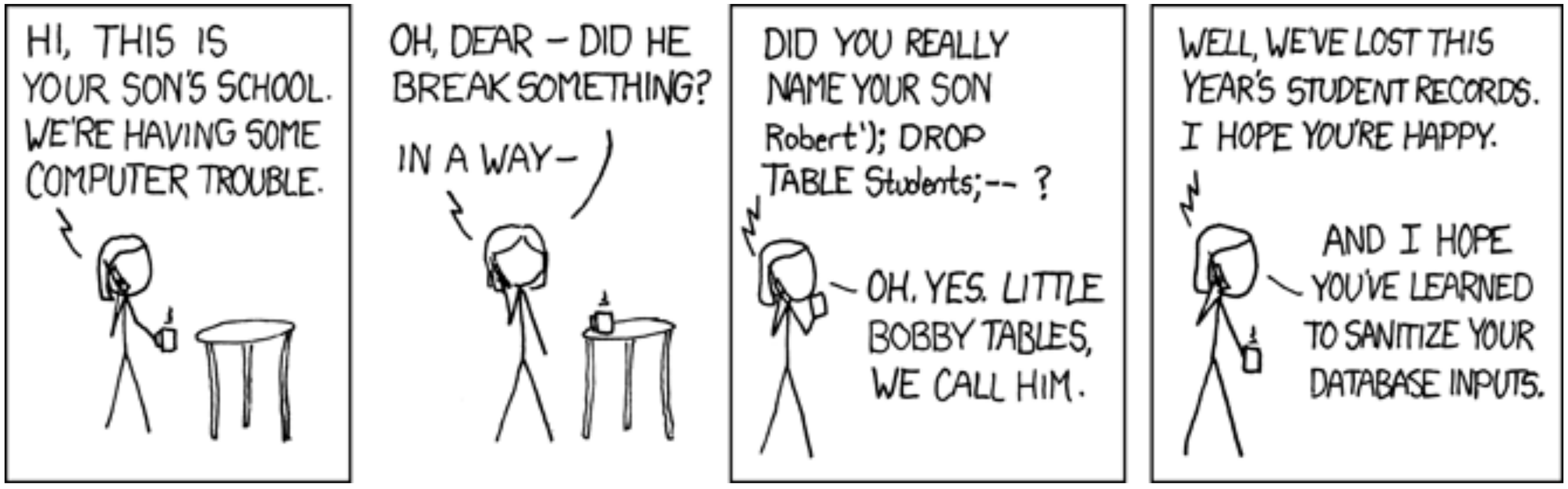

```
CREATE TABLE Works_On
(  Essn              CHAR(15)          NOT NULL,
   Pno               INT               NOT NULL,
   Hours             DECIMAL(3,1)      NOT NULL,
  CONSTRAINT WorksOnPK PRIMARY KEY (Essn, Pno),
  CONSTRAINT WorksEmpFK FOREIGN KEY (Essn)
      REFERENCES Employee(Ssn),
  CONSTRAINT WorksProjFK FOREIGN KEY (Pno)
      REFERENCES Project(Pnumber));
```

Example: Company Database (6)



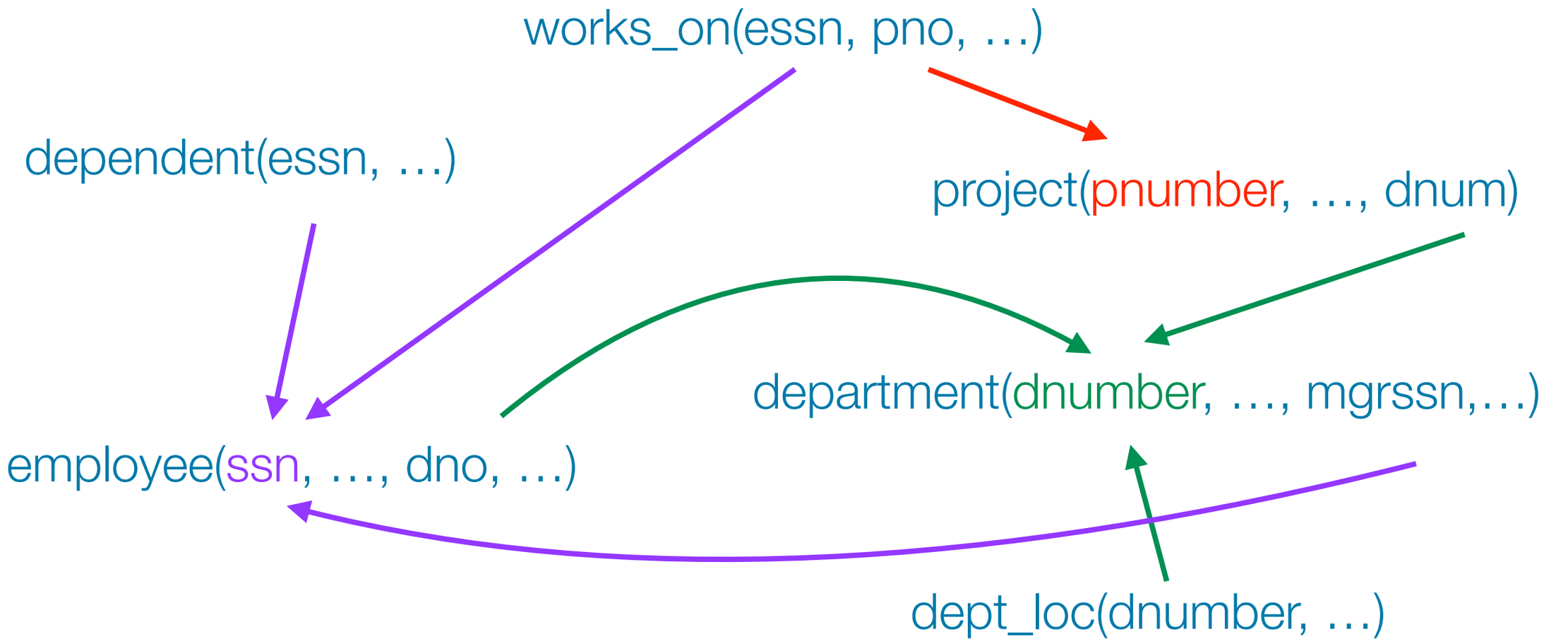

```
CREATE TABLE Dependent
(  Essn              CHAR(9)          NOT NULL,
   Dep_name          VARCHAR(15)      NOT NULL,
   Sex               CHAR,
   Bdate             DATE,
   Relationship      VARCHAR(8),
  CONSTRAINT DepPK PRIMARY KEY (Essn, Dep_name),
  CONSTRAINT DepEmpFK FOREIGN KEY (Essn)
      REFERENCES Employee(Ssn));
```

Drop Tables with Care



https://xkcd.com/327/

“Circular” Integrity Constraints



PROBLEM: Cannot define a referential integrity constraint when the referenced attribute does not exist!

“Circular” Integrity Constraints: Solution

Solution: use ALTER TABLE … ADD CONSTRAINT command after creating the table without referential constraints

```
CREATE TABLE emp1
( ssn CHAR(9),
  dno INT
  CONSTRAINT empPK PRIMARY KEY (ssn));

CREATE TABLE dept1
( dnumber INT,
  mgrssn CHAR(9)
  CONSTRAINT deptPK PRIMARY KEY (dnumber));
```

“Circular” Integrity Constraints: Solution (2)

```
ALTER TABLE emp1 ADD CONSTRAINT empFK
    FOREIGN KEY (dno) REFERENCES dept1(dnumber);

ALTER TABLE dept1 ADD CONSTRAINT deptFK
    FOREIGN KEY (mgrssn) REFERENCES emp1(ssn);
```

It should work, but what about when I insert a tuple?
e.g., INSERT INTO emp1 VALUES(‘44444444’, 12)
Chicken & egg problem all over again!

“Circular” Integrity Constraints: Solution Part II

Solution: use DEFERRED constraints which delays the checking of a constraint until the commit command is issued

```
ALTER TABLE emp1 DROP CONSTRAINT empFK;

ALTER TABLE emp1 ADD CONSTRAINT empFK
    FOREIGN KEY (dno) REFERENCES dept1(dnumber)
    INITIALLY DEFERRED DEFERRABLE;

INSERT INTO emp1 VALUES (‘444444444’, 12);
INSERT INTO dept1 VALUES (12, ‘444444444’);
COMMIT;
```

“Circular” Constraints in MySQL

- All constraints are enforced immediately so there are no deferred constraints

- This solution can not be used in MySQL

- Only solution is to drop the foreign key and avoid having the circular referential constraint

SQL Modifications/Updates

- A modification command does not return a result but it changes the database

- There are 3 kinds of modifications

- **INSERT** tuple(s)

- **DELETE** tuple(s)

- **UPDATE** the value(s) of existing tuples

SQL Modification: INSERT

- Add one more more tuples to an existing relation
- Two forms of **INSERT**:
  - Literal values (constant or known values)
  - Result from a **SELECT** command

SQL Modification: INSERT (2)

Inserting a tuple using literal/constant values
Syntax:
INSERT INTO <table name>[(<attr names>)]
VALUES (<list of values>);

- Complete tuple: omitting [(<attr names>)] means you must specify all attribute values in the exact order defined in relation

- Partial tuple: specify a subset of the attribute values in the same order as the list of attributes [(<attr names>)]

SQL Modification: INSERT (3)

Inserting a tuple using SELECT command

Syntax:
INSERT INTO <table name>[(<attr names>)] (<SELECT subquery>)

- Multiple tuples may be added dependent on the **SELECT** subquery relation

SQL Example: INSERT

- Complete tuple:

  INSERT INTO employee VALUES (‘Joyce’, ‘C’, ‘Ho’, ‘111223333’, ’1985-02-05’, ‘400 Dowman Drive, Atlanta, GA’, ‘F’, ‘150000’, ‘987654321’, 5);

- Partial tuple:

  INSERT INTO employee(fname, lname, ssn) VALUES (‘Joyce’, ‘Ho’, ‘111223333’);

SQL Example: INSERT w/ SELECT

Suppose we want a new table that has the name, number of employees, and total salaries for each department. We first create the table then load it with the information from the database.

```
CREATE TABLE dept_info
( dept_name   VARCHAR(10),
  no_of_emps  INT,
  tot_salary      INT);

INSERT INTO dept_info
          (SELECT       dname, count(*), sum(salary)
           FROM         department, employee
           WHERE        dnumber = dno
           GROUPY BY dname);
```

MySQL: Bulk Import

- All respectable RDBMS provide utilities to import data from text files
  - Syntax for uploading data will vary based on vendor
- MySQL allows the **LOAD DATA INFILE** (http://dev.mysql.com/doc/refman/5.7/en/load-data.html)
  - For a pipe-delimited file (| separates each column):
    LOAD DATA LOCAL INFILE <filename>
    {REPLACE | IGNORE} INTO TABLE <table name>
    FIELDS TERMINATED BY '|';

SQL Modification: DELETE

- Remove tuples from a relation

- Syntax:
  DELETE FROM <relation>
  WHERE <condition>;

- Be careful! All tuples that satisfy the condition clause are deleted

- Tuples are deleted from only one table at a time unless **CASCADE** is specified on a referential integrity constraint

- What happens if we don't specify a **WHERE** clause?

SQL Example: DELETE

Delete all employees with the last name Brown

```
DELETE FROM employee
WHERE  lname = ‘Brown’;
```

SQL Modification: UPDATE

- Modify/change certain attributes in certain tuples of a relation

- Syntax:
  UPDATE <relation>
  SET <list of attribute assignments>
  WHERE <condition>;

- **UPDATE** command modifies tuples in the same relation

SQL Example: UPDATE

Change the location and controlling department number of project 10 to ‘Bellaire’ and 5, respectively.

```
UPDATE project
SET    plocation = ‘Bellaire’, dnum = 5
WHERE  pnumber = 10;
```

SQL Example: UPDATE (2)

Give all employees in the ‘Research’ department a 10% raise

```
UPDATE employee
SET    salary = salary * 1.1
WHERE  dno IN (SELECT dnumber
               FROM   department
               WHERE dname = ‘Research’);
```

- Reference to salary attribute on the right of = refers to the salary value before modification
- Reference to salary attribute on the left of = refers to salary value after modification

SQL Introduction: Recap

- Introduction
- Data Definition
  - Create Database
  - Create Table
- SQL Modification /