# CptS 451- Introduction to Database Systems

# Relational Model and SQL DDL

( DMS Ch-3)

**Instructor: Sakire Arslan Ay**

WASHINGTON STATE UNIVERSITY
*World Class. Face to Face.*

# Outline


- Relational Model
- Constraints on Relations
  - Domain constraints
  - Integrity constraints
    - Key constraints
    - Foreign key constraints
  - Enforcing integrity constraints
- Introduction to SQL

# Relational Model


- Where are we?

# Database Design: A 6-Step Program


1. Requirements Analysis: what data, apps, critical operations
2. Conceptual DB Design: high-level description of data and constraints – typically using ER model
3. Logical DB Design: conversion into a schema
   - pick a type of DBMS, relational DBMS is most popular and is our focus
4. Schema Refinement: normalization (eliminating redundancy)
5. Physical DB Design: consider workloads, indexes and clustering of data
6. Application/Security Design

# ER Model vs. Relational Model


- Both are used to model data
- ER model has many concepts
  - entities, relations, attributes, etc.
  - well-suited for capturing the app. requirements
  - not well-suited for computer implementation (does not even have operations on its structures)
- Relational model
  - just has a single concept: relation (table)
  - world is represented with a collection of tables
  - well-suited for efficient manipulations on computers

# Relational Model


- Main idea:
  - Table: **relation**
  - Table header: **relation schema**
  - Column header: **attribute**
  - Row: **tuple**
- Attributes:
  - Each attribute has a **domain**
  - Each attribute is **atomic**.

Employee

| ssno | name | salary |
|---|---|---|
| 111-11-1111 | John Yates | 80,000 |
| 222-22-2222 | Vasiliy Bunakov | 40,000 |

# Relational Model

Main Concepts:

1. Table: relation
2. Column header: attribute
3. Row: tuple

Relation R:

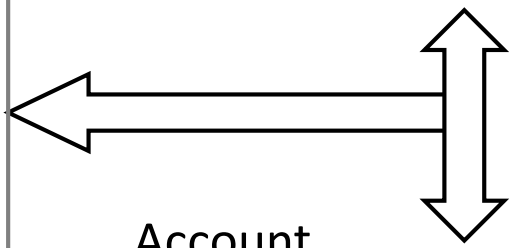
*account* = { (150, 20, 11000),
(160, 23, 2300),
(180, 23, 32000) }

Relation schema: R(attributes)

**Account (AccountId, CustomerId, Balance)**

Account

| AccountId | CustomerId | Balance |
|---|---|---|
| 150 | 20 | 11,000 |
| 160 | 23 | 2,300 |
| 180 | 23 | 32,000 |



Account

| CustomerId | AccountId | Balance |
|---|---|---|
| 23 | 160 | 2,300 |
| 23 | 180 | 32,000 |
| 20 | 150 | 11,000 |

**(No order in attributes or tuples)**

# Relational Model: Some Notation


- The **schema** (or description) of a **relation**:
  - Denoted by **R(A1, A2, .....An)**
  - **R** is the **name of the relation**
  - **A1, A2, ..., An** are the attributes of the relation
  - **R.Ai** refers to **R**'s attribute **Ai**
- A **database schema** consists of
  - A set of **relation schemas**, e.g., S= (R1, R2… Rn)
  - A set of **constraints** over the relation schemas

**CourseOfferings (semester, course#, instructor)**

**CourseOfferings**

| semester | course# | instructor |
|---|---|---|
| Fall, 2013 | EE 214 | Cole |
| Fall, 2013 | Cpts 121 | O'Fallon |
| Fall, 2013 | CptS422 | O'Fallon |
| Spring,2014 | CptS317 | Kalyanaraman |
| Spring, 2014 | CptS460 | Wang |

**CourseOfferings.instructor**

# Relational Model: Some Notation


- **R[Ai]** denote the column of the relation instance that corresponds to attribute **Ai**
  - **R[A1,A2,A3]** refer to the columns for attributes **A1, A2, A3**
  - If the set of all attributes is **{A1,…An}** then, **R[A1,…,An]** refers to the complete table
- Let **t** be a tuple (row) in relation R
  - **t[Ai]** refers to the value of the attribute **Ai** for tuple **t**
  - **t[A1,A2,A3]** refers to the sub-tupple of **t** containing the values of the attributes **A1, A2, A3**

**CourseOfferings [course#, instructor]**

**CourseOfferings**

| semester | course# | instructor |
|---|---|---|
| Fall, 2013 | EE 214 | Cole |
| t → Fall, 2013 | Cpts 121 | O'Fallon |
| Fall, 2013 | CptS422 | O'Fallon |
| Spring,2014 | CptS317 | Kalyanaraman |
| Spring, 2014 | CptS460 | Wang |

**t** = <"Fall,2013" "CptS121", "O'Fallon">
**t**[course#] = "CptS121"
**t**[instructor] = "O'Fallon"

# Attributes & Nulls


- Attributes:
  - Each attribute has a domain
    - The set of allowed values for each attribute
  - Each attribute is atomic
    - We cannot refer to or directly see a subpart of the value
    - Composite values and multi-valued attributes are not allowed
  - The number of attributes is called the degree of the relation

- Attributes can have a special value: **NULL**
  - Can mean not known: we don't know Jack's address
  - Or, does not exist: savings account 1001 does not have "overdraft"

Customer(Id, Name, Addr)

| Id | Name | Addr |
|---|---|---|
| 20 | Jared | Pullman |
| 23 | Jane | Seattle |
| 32 | Jack | **NULL** |

# Integrity Constraints over Relations


- Why?
  - Condition specified on a database schema
  - Enable DBMSs to check that new data satisfies the specified consitions
  - Make application programmers' lives easier
    - If DBMS guarantees account >=0, the debit application programmer need not worry about overdrawn accounts
  - Enable identification of redundancy in schemas
    - Helps in good database design
  - Help the DBMS in query processing
    - Constraints can help the query optimizer choose a good query execution plan (e.g., by helping it with intermediate result size estimation)

# Integrity Constraints over Relations


- Constraint types :
  1. Domain constraints
  2. Key constraints
  3. Entity identity constraints  (disallowing null values)
  4. Foreign key constraints
     - Inclusion dependencies (generalization of foreign key constraints)

# 1. Domain Constraints


- Every attribute has a type (i.e., domain)
  - integer, float, date, boolean, string, etc.
- An attribute can have a domain constraint, e.g.:
  - Id > 0
  - Salary > 0.0
  - Age < 100
  - City in {Portland, Seattle, Pullman}
- An insertion can try to violate the domain constraint
  - DBMS checks if insertion violates domain constraint and rejects the insertion

| Integer | String | String |
|---|---|---|
| **Id** | **Name** | **City** |
| 20 | Tom | Seattle |
| 23 | Jane | **San Diego** |
| -2 | Jack | Pullman |

**violations**

# 2. Key Constraints


## Keys of Relations:

- Definition:
  - A **minimal** subset of the attributes that uniquely identifies a tuple.
- Logical statement:
  - Given
    - Relation schema: **R =(A1, A2,…,An)**
    - Set of attributes: **{A1,A2,…,An}**
    - A subset of attributes: **K ⊆ {A1,A2,…,An}**
    - Relational tuples: **T={t1,…,tm}**
  - For any two distinct tuples **t1∈T** and **t2∈T, if** the relation **R** has the constraint, **t1[K] ≠ t2[K]**, then **K** is a key of the relation schema **R**

# 2. Key Constraints (cont.)


- Example:

Log1(LogId, AccountID, Transact#, Time, Amount)

| LogID | AccountID | Transact# | Time | Amount |
|---|---|---|---|---|
| 1001 | 111 | 4 | 1/12/02 12:00:01 PM | $100 |
| 1002 | 122 | 4 | 12/28/01 2:23:30 PM | $20 |
| 1003 | 333 | 6 | 9/1/00 5:03:00 PM | $60 |

Log2(AccountID, Transact#, Time, Amount)

| AccountID | Transact# | Time | Amount |
|---|---|---|---|
| 111 | 4 | 1/12/02 12:00:01 PM | $100 |
| 122 | 4 | 12/28/01 2:23:30 PM | $20 |
| 333 | 6 | 9/1/00 5:03:00 PM | $60 |

- Features:
  - There are usually multiple *keys.* The set of all keys for a relation is called the *candidate keys.*
  - All attributes always form a *key* (Note that the relational model doesn't allow duplicates)
  - Minimal key (no proper subset is a key)
  - If more than one key: choose *one (the minimal)* as the *primary key*

# 3. Entity Identity Constraints: Disallowing Null Values

- Some fields are too important to contain null values
  - E.g., **Sales(saleID, customer, salesman, date, amount)**
    - Here, we do not want 'customer' to contain a null value
- A primary key **cannot** contain a null value
  - Otherwise, we are not able to differentiate tuples

**Sales**

| saleID | Customer | Salesman | Date | Amount |
|---|---|---|---|---|
| 112587 | Tom | Cindy | 1/21/2015 | 2 |
| 112588 | NULL | Cindy | NULL | 3 |
| NULL | NULL | Jack | 1/11/2015 | 4 |

We don't want these! (Solution: Disallow NULLs for this attribute)

# 4. Foreign Key Constraints


- Specified between two relations to maintain the correspondence between tuples in these relations

Student

| Id | Name | Dept |
|---|---|---|
| 1111 | Mike | EECS |
| 2222 | Harry | MME |
| 3333 | Ford | EECS |

Department

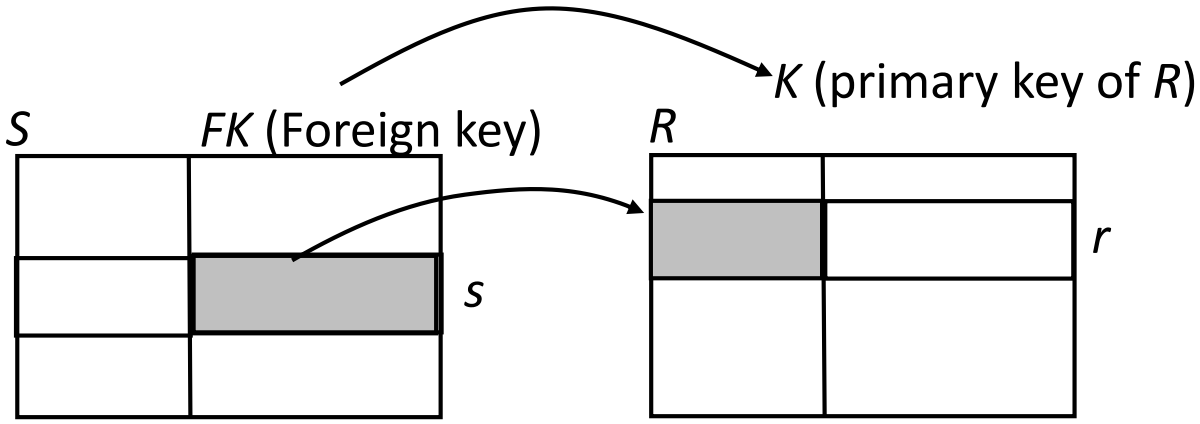
| Name | chair |
|---|---|
| **EECS** | Tom |
| **MME** | Jane |
| MATH | Jack |

***Student.Dept to Dept.Name***

(Every value of ***Student.Dept*** must also be a value of ***Dept.Name***)

# Foreign Key Constraints (cont.)

- **Definition** (*foreign key):*
  - Relation schemas: **R** and **S**.
  - **R** has a primary key ***K*** (a set of attributes)
  - A set of attributes ***FK*** is a *foreign key* of **S** if:
    - The attributes in **FK** have same domains as the attributes (or correspond to the attributes) in **K**
    - The values for **FK** in each tuple **s** in **S** either occur as values of **K** of a tuple in **r** of **R** or else **FK** is NULL
- ***FK*** is a foreign key that refers to the primary key ***K*** of **R**



# Another Example


Account

| AccountId | CustomerId | Balance |
|---|---|---|
| 150 | 20 | 11,000 |
| 160 | 23 | 2,300 |
| 180 | 23 | 32,000 |

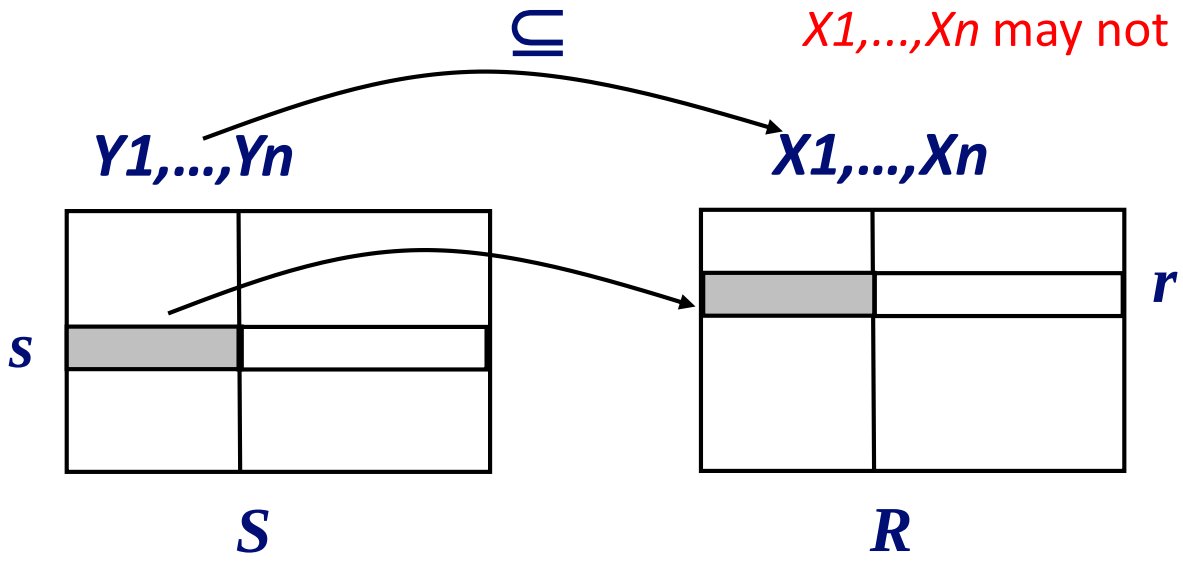
Customer

| Id | Name | Addr |
|---|---|---|
| 20 | Tom | Pullman |
| 23 | Jane | Seattle |
| 32 | Jack | Portland |

***Account.CustomerId*** *to* ***Customer.Id***

# Inclusion Dependencies

- Generalization of foreign key constraints
  - Inclusion dependency **S[Y1,…,Yn] ⊆ R[X1,…,Xn]** means: *values in relation **S** refer to values in relation **R***
  - *Foreign key is an inclusion dependency in which {X1,…,Xn} is the primary key of R*



# Data Modifications and Constraints


- Modification, insertion, and deletion requests can lead to violations of one or more integrity constraints
- DBMS must check for violations at the end of each operation and suitably allow or disallow the operation
- Impacts of data modifications on foreign key constraints can be sometimes tricky to handle!

# Example: Delete from *Referenced* Table


**Takes**

| student# | semester | course# |
|---|---|---|
| 1111 | Fall, 2013 | CptS422 |
| 2222 | Spring,2014 | CptS317 |
| 3333 | Spring, 2014 | CptS317 |

**CourseOfferings**

| semester | course# | instructor |
|---|---|---|
| Fall, 2013 | CptS422 | O'Fallon |
| ~~Spring,2014~~ | ~~CptS317~~ | ~~Kalyanaraman~~ |
| Fall, 2013 | CptS460 | Wang |

(semester,course#) is the primary key

- Foreign key constraint:
  - Takes(semester, course#) refers to CourseOfferings(semester, course#)
- Deleting a course: "Spring, 2014, CptS317" from "CourseOfferings" relation.
- What should happen to any tuples in "Takes" that refer to "Spring, 2014, CptS317"?

# Integrity Maintenance Policies


1. **Abort**: reject the update (sometimes called Restrict)
2. **Cascade**: delete tuples from **Takes** that refer to "Spring,2014 CptS317"
   - Notice that deleting this Takes record could have a cascading effect, since other records may refer to it
3. **Set NULL**: set referring values in Takes to NULL

deleted

| student# | semester | course# |
|---|---|---|
| 1111 | Fall, 2013 | CptS422 |
| ~~2222~~ | ~~Spring,2014~~ | ~~CptS317~~ |
| ~~3333~~ | ~~Spring, 2014~~ | ~~CptS317~~ |

| semester | course# | instructor |
|---|---|---|
| Fall, 2013 | CptS422 | O'Fallon |
| ~~Spring,2014~~ | ~~CptS317~~ | ~~Kalyanaraman~~ |
| Fall, 2013 | CptS460 | Wang |

Set NULL

| student# | semester | course# |
|---|---|---|
| 1111 | Fall, 2013 | CptS422 |
| 2222 | NULL | NULL |
| 3333 | NULL | NULL |

| semester | course# | instructor |
|---|---|---|
| Fall, 2013 | CptS422 | O'Fallon |
| ~~Spring,2014~~ | ~~CptS317~~ | ~~Kalyanaraman~~ |
| Fall, 2013 | CptS460 | Wang |

**Takes** **CourseOfferings**

# Example: Update *Referenced* Table


Takes

| student# | semester | course# |
|---|---|---|
| 1111 | Fall, 2013 | CptS422 |
| 2222 | Spring,2014 | ~~CptS317~~ CptS417 |
| 3333 | Spring, 2014 | ~~CptS317~~ CptS417 |

CourseOfferings

| semester | course# | instructor |
|---|---|---|
| Fall, 2013 | CptS422 | O'Fallon |
| Spring,2014 | ~~CptS317~~ CptS417 | Kalyanaraman |
| Fall, 2013 | CptS460 | Wang |

(semester,course#) is the primary key

| student# | semester | course# |
|---|---|---|
| 1111 | Fall, 2013 | CptS422 |
| 2222 | NULL | NULL |
| 3333 | NULL | NULL |

| semester | course# | instructor |
|---|---|---|
| Fall, 2013 | CptS422 | O'Fallon |
| Spring,2014 | ~~CptS417~~ CptS417 | Kalyanaraman |
| Fall, 2013 | CptS460 | Wang |

- In CourseOfferings, change "Spring 2014, CptS317" to "Spring 2014, CptS417"
- Solutions:
  - **Abort (or Restrict)**: reject the update
  - **Cascade**: In Takes, also change all appearances of "Spring 2014, CptS317" to "Spring 2014, CptS417"
  - **Set NULL**: set the referencing value(s) to null

# Example: Update *Referencing* Table


Takes

| student# | semester | course# |
|---|---|---|
| 1111 | Fall, 2013 | ~~CptS422~~ CptS444 |
| 2222 | Spring,2014 | CptS317 |
| 3333 | Spring, 2014 | CptS317 |

CourseOfferings

| semester | course# | instructor |
|---|---|---|
| Fall, 2013 | CptS422 | O'Fallon |
| Spring,2014 | CptS317 | Kalyanaraman |
| Fall, 2013 | CptS460 | Wang |

**illegal**

| student# | semester | course# |
|---|---|---|
| 1111 | Fall, 2013 | ~~CptS422~~ CptS460 |
| 2222 | Spring,2014 | CptS317 |
| 3333 | Spring, 2014 | CptS317 |

| semester | course# | instructor |
|---|---|---|
| Fall, 2013 | CptS422 | O'Fallon |
| Spring,2014 | CptS317 | Kalyanaraman |
| Fall, 2013 | CptS460 | Wang |

**legal**

- In Takes, change "Fall 2013, CptS422" to a new value
- We need to make sure that the new value belongs to the values of CourseOfferings
- If it doesn't, the update should be rejected

# Example: Insert to *Referencing* Table


Takes

| student# | semester | course# |
|---|---|---|
| 1111 | Fall, 2013 | CptS422 |
| 2222 | Spring,2014 | CptS317 |
| 3333 | Spring, 2014 | CptS317 |
| **4444** | **Fall, 2014** | **CptS422** |

CourseOfferings

| semester | course# | instructor |
|---|---|---|
| **Fall, 2013** | **CptS422** | O'Fallon |
| Spring,2014 | CptS317 | Kalyanaraman |
| Fall, 2013 | CptS460 | Wang |

- In Takes, insert tuple "4444,Fall 2013, CptS422"
- We need to make sure that the new value for (semester,course#) appears in CourseOfferings
- If it doesn't, the insert should be rejected


# Introduction to SQL

# SQL - Historical Perspective


- Developed by IBM in mid 70s
- First standardized in 1986 by ANSI --- SQL1
- Revised in 1992 --- SQL2. Approximate 580 pages describing syntax and semantics
- In 1999, ANSI/ISO released the *SQL3*. Many additions for:
  - support for multimedia data
  - addition of abstract data types and object-orientation
  - support for calling programmed functions from within SQL
- Every vendor has a slightly different version of SQL
- If you ignore the details, basic SQL is very simple and declarative. Hence easy to use.

# Why SQL?


- SQL is a declarative language.
  - Say "what to do" rather than "how to do it."
  - Avoid a lot of data-manipulation details needed in procedural languages like C++ or Java.
- Database management system figures out "best" way to execute query.
  - Called "query optimization."

# SQL in Different Roles


- Data definition Language (DDL):
  - allows users to create the relations and constraints.
- Data Manipulation Language (DML)
  - Query Language:
    - declarative -- you specify what you want and not how to retrieve, easy to learn and program
  - Updates:
    - insert, delete, and update tables
- Other commands:
  - View definition language:
    - commands to define rules
    - updates through views generally not supported
  - Transaction Control:
    - commands to specify beginning and end of transactions.

# SQL (cont)


- SQL statements can be run interactively via GUI or prompt.
- Embedded SQL:
  - has been embedded in a variety of host languages:
    - C, C++, PERL, Smalltalk, Java (vendor dependent)

# Data in SQL


1. Atomic types, a.k.a. data types
2. Tables built from atomic types
3. No nested tables, only flat tables are allowed!
   - We will see later how to decompose complex structures into multiple flat tables

## Data Types in SQL

- Characters:
  - CHAR(20) -- fixed length
  - VARCHAR(40) -- variable length
- Numbers:
  - BIGINT, INT, SMALLINT, TINYINT
  - REAL, FLOAT -- differ in precision
  - MONEY
- Times and dates:
  - DATE
  - TIME
  - DATETIME -- SQL Server
- Others...

# SQL as DDL (Data Definition Language)


- **Simple Data Declarations**

```
CREATE TABLE Dept
(dno        int  NOT NULL,
 dname      varchar(30),
 mgr        varchar(20)
);

CREATE TABLE Emp
 (ename     char(20)   NOT NULL,
  dno       int DEFAULT 0,
  sal       int
 );
```

Don't allow null values

Default value is 0

Dept

| dno | Dname | Mgr |
|---|---|---|
| 2752 | EECS | Behrooz Shirazi |
| 2534 | MME | Michael Kessler |
| 2651 | MATH | NULL |

Emp

| ename | dno | sal |
|---|---|---|
| John Yates | 2752 | 81,000 |
| Michael Kessler | 2534 | 100,500 |
| John Lee | 0 | 32,000 |

# Define New Domains


```
CREATE DOMAIN personDom CHAR(20);

CREATE TABLE emp
 ( ename        personDom,
   dno          int default 0,
   sal          real
  );
```

# Creating and Modifying Relations using SQL


**CREATE TABLE R** (
    **A1 D1** *[not null] [default V1] [primary key] [unique],*
    …
    **An Dn** *[not null] [default Vn] [primary key] [unique],*
    *<integrity constraint 1>* ,
    …
    *<integrity constraint k>*
)

**Each integrity constraints could be:**

1. primary key
2. UNIQUE
3. "check(predicate)": specifies condition to be satisfied by each tuple
4. foreign key

# 1. Declaring Keys – Primary Key


There are 2 ways to declare the primary key:

**1)** When the attribute is listed in the relation schema.

```
CREATE TABLE emp
( ssn       int Primary Key,
  name      char(15),
  dno       int,
);
```

**2)** As an additional declaration that lists the attribute(s) that form the key.

```
CREATE TABLE emp
( ssn       int,
  name      char(15),
  dno       int,
  Primary Key (ssn)
);
```

```
CREATE TABLE emp
( ssn       int,
  name      char(15),
  dno       int,
  Primary Key (dno,name)
);
```

Need to use the second method if the key consists of multiple attributes

# 2. UNIQUE Constraint


- Alternative key declaration with UNIQUE

```
CREATE TABLE emp
( SSN       int  Primay Key,
  eID       int UNIQUE,
  name      char(15),
  dno       int,
);
```

```
CREATE TABLE emp
( SSN       int  Primary Key,
  name      char(15),
  dno       int,
 UNIQUE (dno,name)
);
```

- A table has only one "primary key" in a table, but many "unique"s
- "Unique" may have NULLs.

# 3. CHECK Clause


- Enforces a predicate (condition)

```
CREATE TABLE emp
( SSN       int,
  name      varchar(25),
  dno       int,
  salary     bigint,
  CHECK (salary>=0)
);
```

Ensure that the values of the assets are non-negative

- Can be named (useful to indicate which constraint an update violated)

```
CONSTRAINT  salary_nonzero  CHECK (salary>=0)
```

# 4. Foreign-Key Constraint


There are 2 ways to declare the foreign key:

**1)**

```
CREATE TABLE emp
( SSN  int,
  name  char(15),
  dno  int REFERENCES dept(dept#)
);
```

**2)**

```
CREATE TABLE emp
( ssn     int,
  name  char(15),
  dno    int,
  FOREIGN KEY (dno) REFERENCES dept(dept#)
);
```

- Allow multiple attributes in one foreign-key constraint.
- Allow multiple foreign-key constraints in one table.

emp (SSN, name, dno)

| SSN | name | dno |
|---|---|---|
| 9999 | Jack | 111 |
| 8888 | Alice | 111 |
| 7777 | Lisa | 222 |
| 6666 | Tom | 333 |
| 5555 | John | 333 |

dept(dept#, dname, mgr)

| dept# | dname | mgr |
|---|---|---|
| 111 | Marketing | 9999 |
| 222 | HR | 7777 |
| 7777 | IT | 5555 |

```
CREATE TABLE dept
( dept# int  PRIMARY KEY,
  dname  char(15),
  mgr  char(15) REFERENCES  emp(SSN)
)
```

# 4. Foreign-Key Constraints (cont.)


- Also called "referential integrity"
- Within an attribute:
  - REFERENCES <TABLE> (<attributes>)
- Separate declaration:
  - FOREIGN KEY <attributes> REFERENCES <table> (<attributes>)
- NULL values allowed for attributes in a foreign key.
  - A foreign-key constraint is automatically satisfied if even one attribute in the foreign key is null.

# Giving Names to Constraints


- Keyword **`CONSTRAINT`**
  - Name a constraint
  - Useful for later altering

```
CONSTRAINT FK1  FOREIGN KEY (dno) REFERENCES dept(dept#));
```

# Enforcing Foreign-Key Constraints


emp (SSN, name, dno)

| SSN | name | dno |
|---|---|---|
| 9999 | Jack | 111 |
| 8888 | Alice | 111 |
| 7777 | Lisa | 222 |
| 6666 | Tom | 333 |
| 5555 | John | ~~333~~ 444 |

dept(dept#, dname, mgr)

| dept# | dname | mgr |
|---|---|---|
| 111 | Marketing | 9999 |
| 222 | HR | 7777 |
| 333 | Operations | 6666 |
| 777 | IT | 8888 |

emp.dno references dept.dept#

- Modification (insert, update) of ***emp***
  - If the new tuple's dno does not exist in dept.dept# , then REJECT!

# Enforcing Foreign-Key Constraints


emp (SSN, name, dno)

| SSN | name | dno |
|---|---|---|
| 9999 | Jack | 111 |
| 8888 | Alice | 111 |
| 7777 | Lisa | 222 |
| 6666 | Tom | 333 |
| 5555 | John | 333 |

dept(dept#, dname, mgr)

| dept# | dname | mgr |
|---|---|---|
| ~~111~~ | ~~Marketing~~ | ~~9999~~ |
| 222 | HR | 7777 |
| 333 | Operations | 6666 |
| 777 | IT | 8888 |

emp.dno references dept.dept#

- Modification (delete, update) of ***dept*** whose old "dept#" is referenced by a record in *emp*. There are 3 strategies:
  - Default: reject
  - Cascade: change the *emp* tuple correspondingly
    - Delete in *dept*: delete the referring record(s) in *emp*
    - Update in *dept*: update the dno of the referring record(s) in *emp* to the new dno
  - Set Null: change dno value in referring record(s) in *emp* to NULL

# Foreign Key Constraint – Choosing a Policy


- Add "**ON** [DELETE,UPDATE] [CASCADE, SET NULL, NO ACTION]" when declaring a foreign key
  - If no policy is specified, the default (NO ACTION) is used.
- Which policy to choose depends on the application.

```
CREATE TABLE emp
(  ssn  INT,
   name  VARCHAR(20),
   dno  INT,
   FOREIGN KEY dno REFERENCES dept(dept#)
   ON DELETE SET NULL
   ON UPDATE CASCADE
);
```

# SQL as DDL - DROP TABLE command


- Used to remove a relation & its *definition*
  - The relation can no longer be used in queries, updates, or any other commands since its description no longer exists

```
DROP TABLE emp;
```

# SQL as DDL - ALTER TABLE command


- Used to add/drop attributes from a relation
- Add attribute:

```
ALTER TABLE relationName ADD attribName attribDomain
```

  - All tuples in the relation are assigned *null as the default value of the* new attribute

- Drop attribute:

```
ALTER TABLE relationName DROP attribName
```

  - Dropping of attributes not supported by many DBMSs

# SQL as DDL - ALTER TABLE command (cont.)


- Since new attribute will have NULL values right after the ALTER command is executed, the NOT NULL constraint is not allowed for such an attribute
- Example:

  **ALTER TABLE** emp **ADD** salary bigint

  - The database users must still enter a value for the new attribute "salary" for each EMPLOYEE tuple. This can be done using the UPDATE command. (We will cover this later)