CS 4604: Introduction to Database Management Systems

SQL I

Virginia Tech CS 4604 Sprint 2021

Instructor: Yinlin Chen

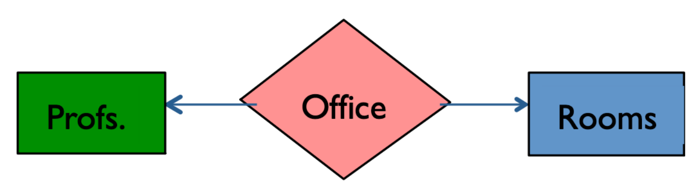
Today’s Topics

- Structured Query Language (SQL)
  - Pronounced ‘Sequel’
  - The most widely used relational query language

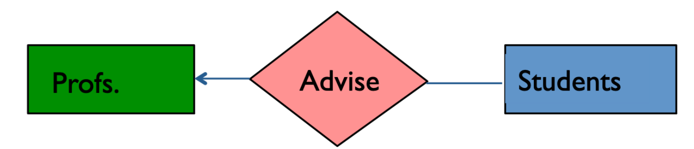
Recap: Cardinality

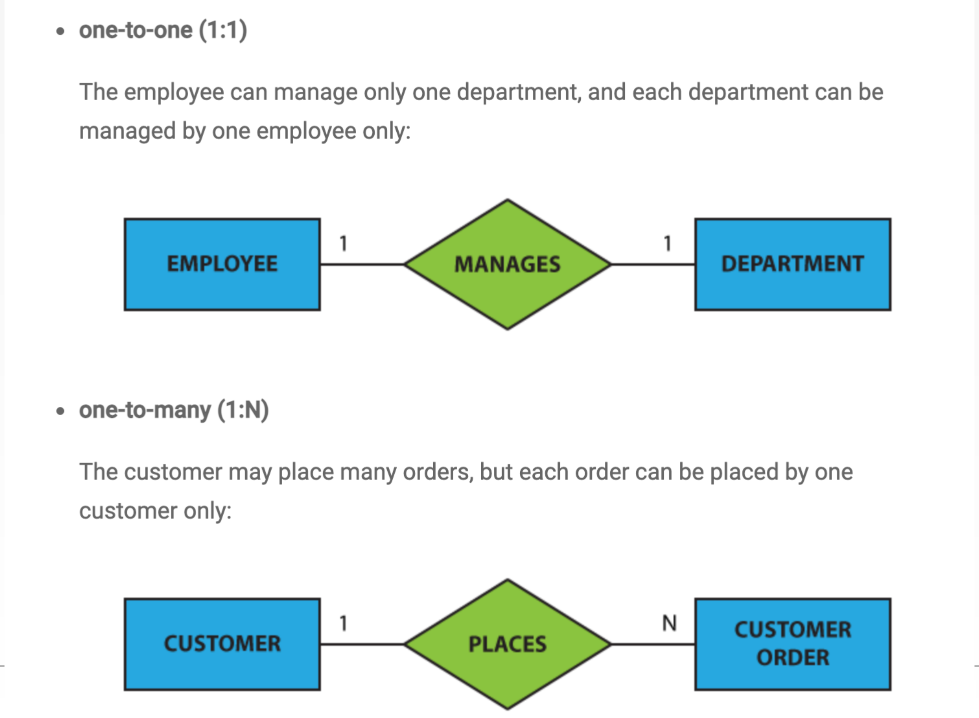
- Peter Chen, the father of ER modeling
- The degree of relationship (cardinality) is represented by characters “1”, “N” or “M” usually placed at the ends of the relationships.
- Chen's notation

one-one



one-many





- **one-to-one (1:1)**

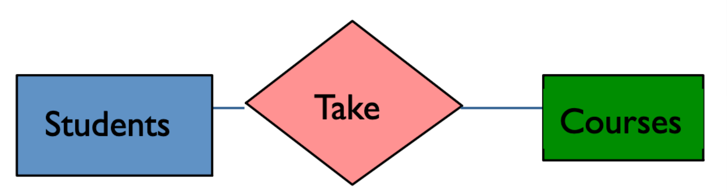
The employee can manage only one department, and each department can be managed by one employee only:

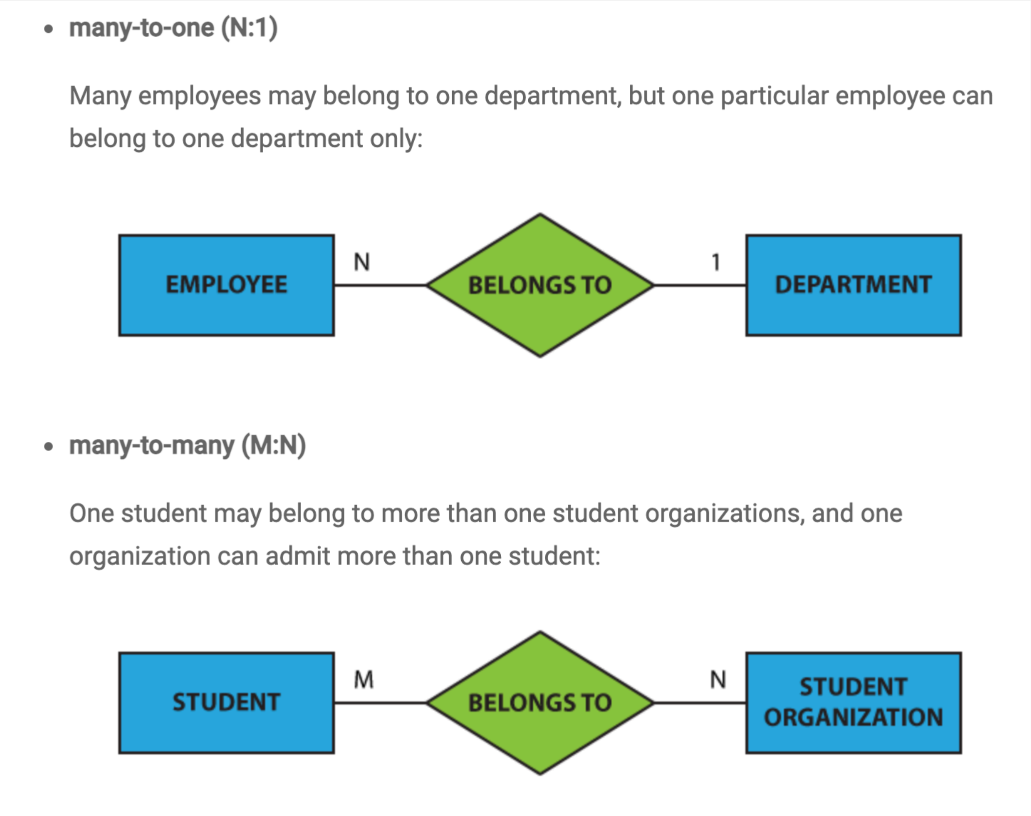
- **one-to-many (1:N)**

The customer may place many orders, but each order can be placed by one customer only:

Recap: Cardinality

many-many





- **many-to-one (N:1)**

  Many employees may belong to one department, but one particular employee can belong to one department only:

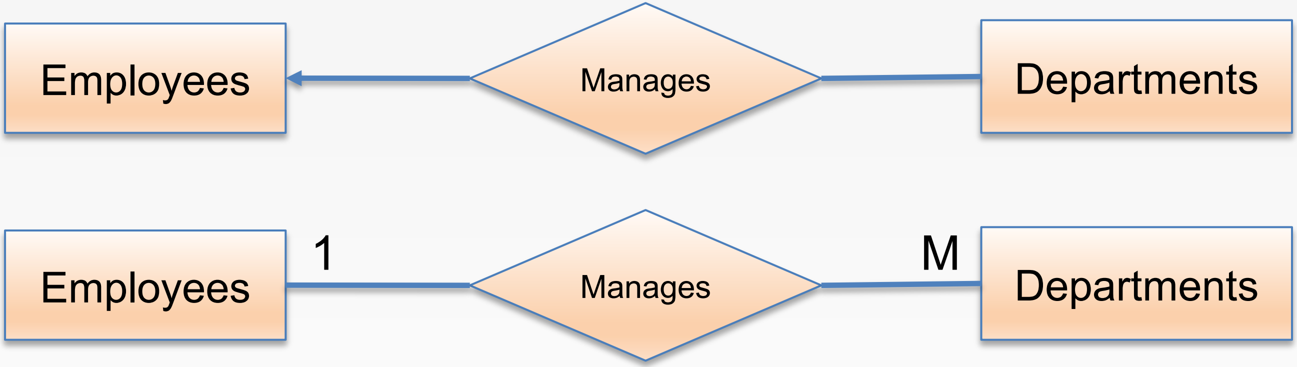
- **many-to-many (M:N)**

  One student may belong to more than one student organizations, and one organization can admit more than one student:

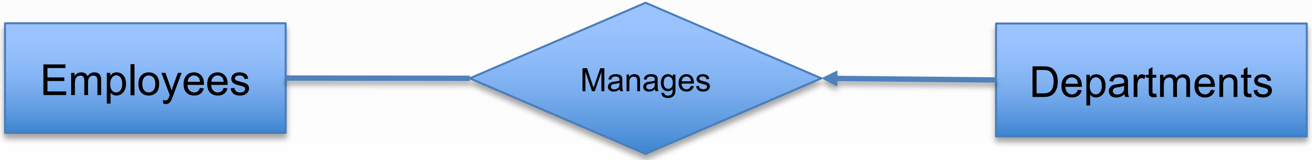
One to Many

- Department has at most one manager. A single Employee is allowed to manage more than one department

In Slide



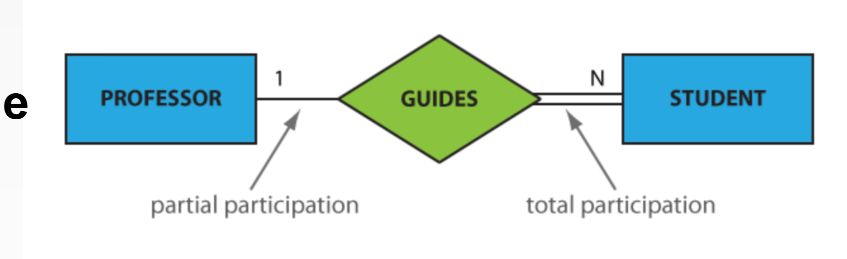
In TextBook

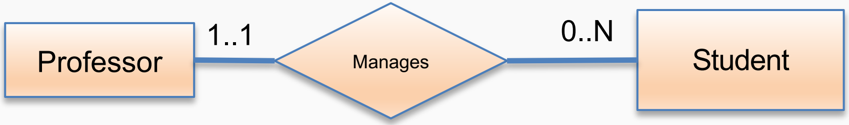


Participation Constraints

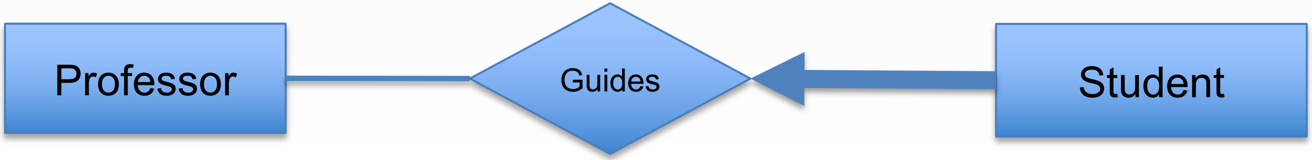
- **Total participation** means that every entity in the set is involved in the relationship, e.g., each student must be guided by a professor (there are no students who are not guided by any professor)
- **Partial participation** means that not all entities in the set are involved in the relationship, e.g., not every professor guides a student (there are professors who don’t).

In Slide

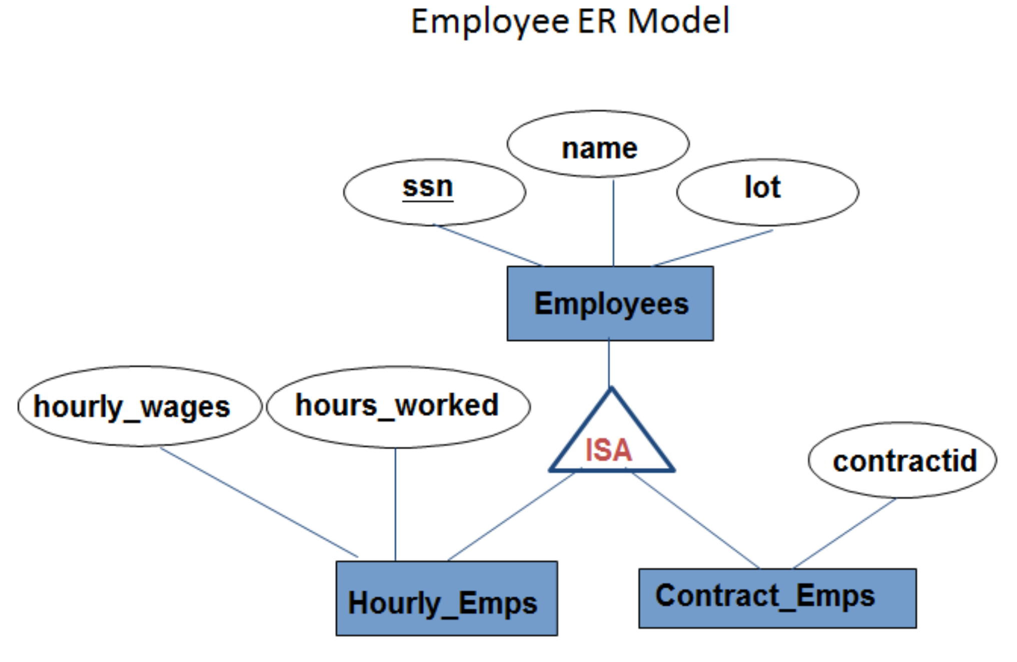




In TextBook



Recap: Class Hierarchies



- **Overlap constraints**: whether two subclasses are allowed to contain the same entity
- **Covering constraints**: whether the entities in the subclasses collectively include all entities in the superclass

RDBMS and SQL

- The DBMS is responsible for efficient evaluation
  - Choose and run algorithms for declarative queries
  - Choice of algorithm must not affect query answer
  - Query optimizer: re-orders operations, generates query plan, and still ensure that the answer does not change
- Many ways to write a query. DBMS figures out a fast way to execute a query, regardless of how it is written

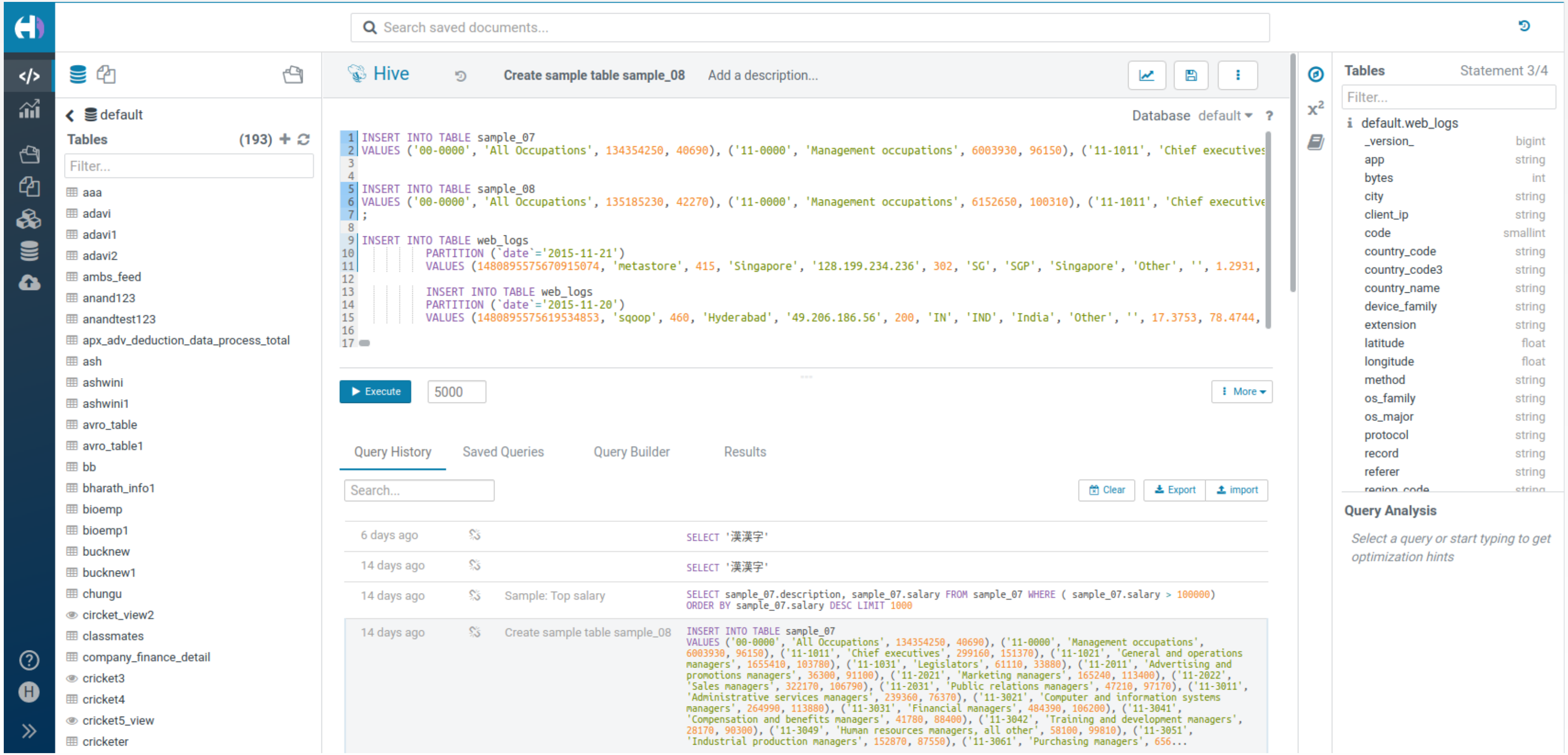
The SQL Query Language

- First version, SQL-86 in 1986, most recent version in 2011 (SQL:2016)
- Accepted by the American National Standards Institute (ANSI) in 1986 and by the International Organization for Standardization (ISO) in 1987
- **Each vendor provides its own implementation** (also called SQL dialect) **of SQL**

Key Characteristics of SQL

- Set-oriented and **declarative**
- Free-form **language**
- Case insensitive
- Can be used both interactively from a command prompt or executed by a program

Using Command Prompt



Tables (193)

aaa
adavi
adavi1
adavi2
ambs_feed
anand123
anandtest123
apx_adv_deduction_data_process_total
ash
ashwini
ashwini1
avro_table
avro_table1
bb
bharath_info1
bioemp
bioemp1
bucknew
bucknew1
chungu
circket_view2
classmates
company_finance_detail
cricket3
cricket4
cricket5_view
cricketer

Create sample table sample_08

Database default

```
INSERT INTO TABLE sample_07
VALUES ('00-0000', 'All Occupations', 134354250, 40690), ('11-0000', 'Management occupations', 6003930, 96150), ('11-1011', 'Chief executives


INSERT INTO TABLE sample_08
VALUES ('00-0000', 'All Occupations', 135185230, 42270), ('11-0000', 'Management occupations', 6152650, 100310), ('11-1011', 'Chief executive
;

INSERT INTO TABLE web_logs
        PARTITION (`date`='2015-11-21')
        VALUES (1480895575670915074, 'metastore', 415, 'Singapore', '128.199.234.236', 302, 'SG', 'SGP', 'Singapore', 'Other', '', 1.2931,

        INSERT INTO TABLE web_logs
        PARTITION (`date`='2015-11-20')
        VALUES (1480895575619534853, 'sqoop', 460, 'Hyderabad', '49.206.186.56', 200, 'IN', 'IND', 'India', 'Other', '', 17.3753, 78.4744,

```

6 days ago		SELECT '漢漢字'
14 days ago		SELECT '漢漢字'
14 days ago	Sample: Top salary	SELECT sample_07.description, sample_07.salary FROM sample_07 WHERE (sample_07.salary > 100000) ORDER BY sample_07.salary DESC LIMIT 1000
14 days ago	Create sample table sample_08	INSERT INTO TABLE sample_07 VALUES ('00-0000', 'All Occupations', 134354250, 40690), ('11-0000', 'Management occupations', 6003930, 96150), ('11-1011', 'Chief executives', 299160, 151370), ('11-1021', 'General and operations managers', 1655410, 103780), ('11-1031', 'Legislators', 61110, 33880), ('11-2011', 'Advertising and promotions managers', 36300, 91100), ('11-2021', 'Marketing managers', 165240, 113400), ('11-2022', 'Sales managers', 322170, 106790), ('11-2031', 'Public relations managers', 47210, 97170), ('11-3011', 'Administrative services managers', 239360, 76370), ('11-3021', 'Computer and information systems managers', 264990, 113880), ('11-3031', 'Financial managers', 484390, 106200), ('11-3041', 'Compensation and benefits managers', 41780, 88400), ('11-3042', 'Training and development managers', 28170, 90300), ('11-3049', 'Human resources managers, all other', 58100, 99810), ('11-3051', 'Industrial production managers', 152870, 87550), ('11-3061', 'Purchasing managers', 656...

Tables Statement 3/4

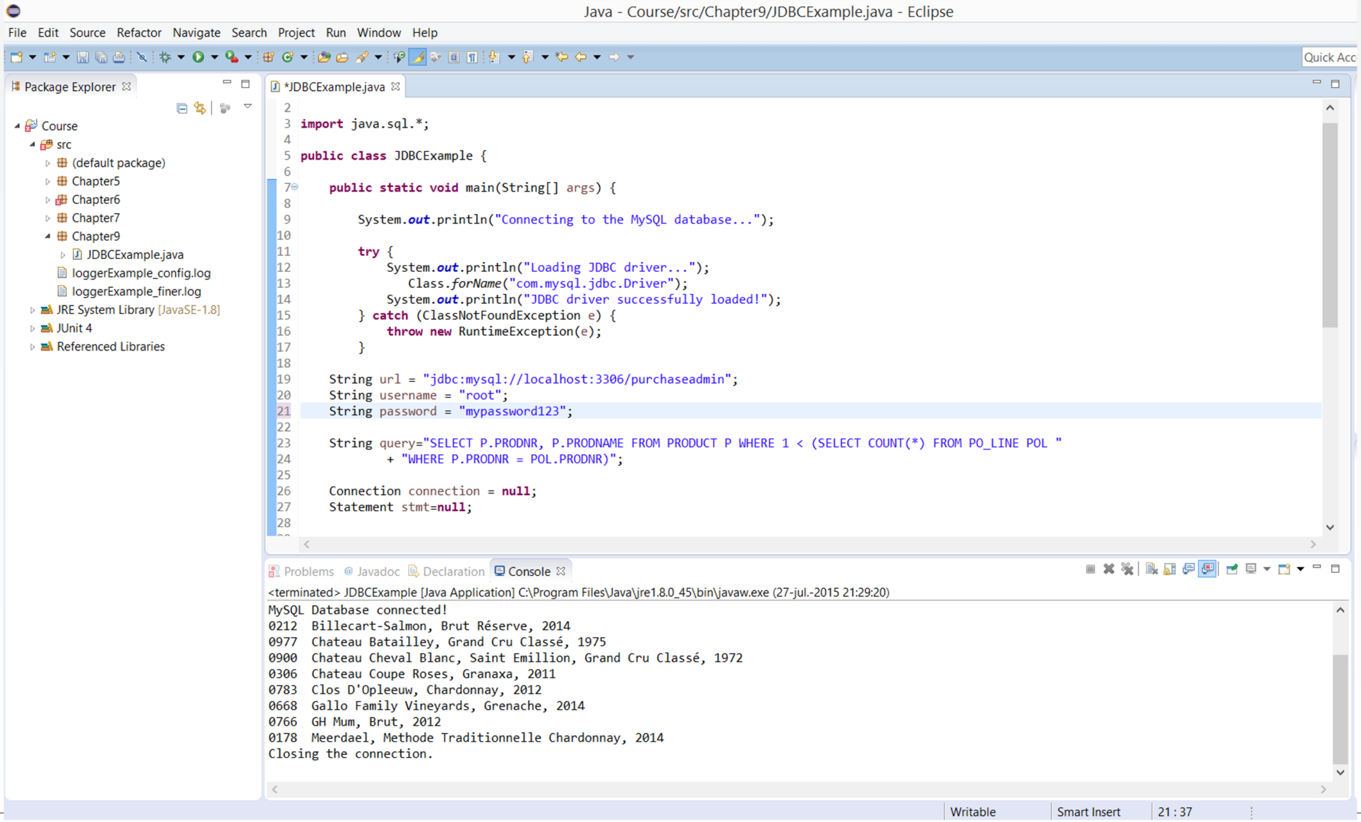
default.web_logs

version	bigint
app	string
bytes	int
city	string
client_ip	string
code	smallint
country_code	string
country_code3	string
country_name	string
device_family	string
extension	string
latitude	float
longitude	float
method	string
os_family	string
os_major	string
protocol	string
record	string
referer	string
region_code	string

Query Analysis

Select a query or start typing to get optimization hints

Executed by a program



```java

import java.sql.*;

public class JDBCExample {

    public static void main(String[] args) {

        System.out.println("Connecting to the MySQL database...");

        try {
            System.out.println("Loading JDBC driver...");
               Class.forName("com.mysql.jdbc.Driver");
            System.out.println("JDBC driver successfully loaded!");
        } catch (ClassNotFoundException e) {
            throw new RuntimeException(e);
        }

    String url = "jdbc:mysql://localhost:3306/purchaseadmin";
    String username = "root";
    String password = "mypassword123";

    String query="SELECT P.PRODNR, P.PRODNAME FROM PRODUCT P WHERE 1 < (SELECT COUNT(*) FROM PO_LINE POL "
            + "WHERE P.PRODNR = POL.PRODNR)";

    Connection connection = null;
    Statement stmt=null;

```

```
<terminated> JDBCExample [Java Application] C:\Program Files\Java\jre1.8.0_45\bin\javaw.exe (27-jul.-2015 21:29:20)
MySQL Database connected!
0212  Billecart-Salmon, Brut Réserve, 2014
0977  Chateau Batailley, Grand Cru Classé, 1975
0900  Chateau Cheval Blanc, Saint Emillion, Grand Cru Classé, 1972
0306  Chateau Coupe Roses, Granaxa, 2011
0783  Clos D'Opleeuw, Chardonnay, 2012
0668  Gallo Family Vineyards, Grenache, 2014
0766  GH Mum, Brut, 2012
0178  Meerdael, Methode Traditionnelle Chardonnay, 2014
Closing the connection.
```

SQL Overview

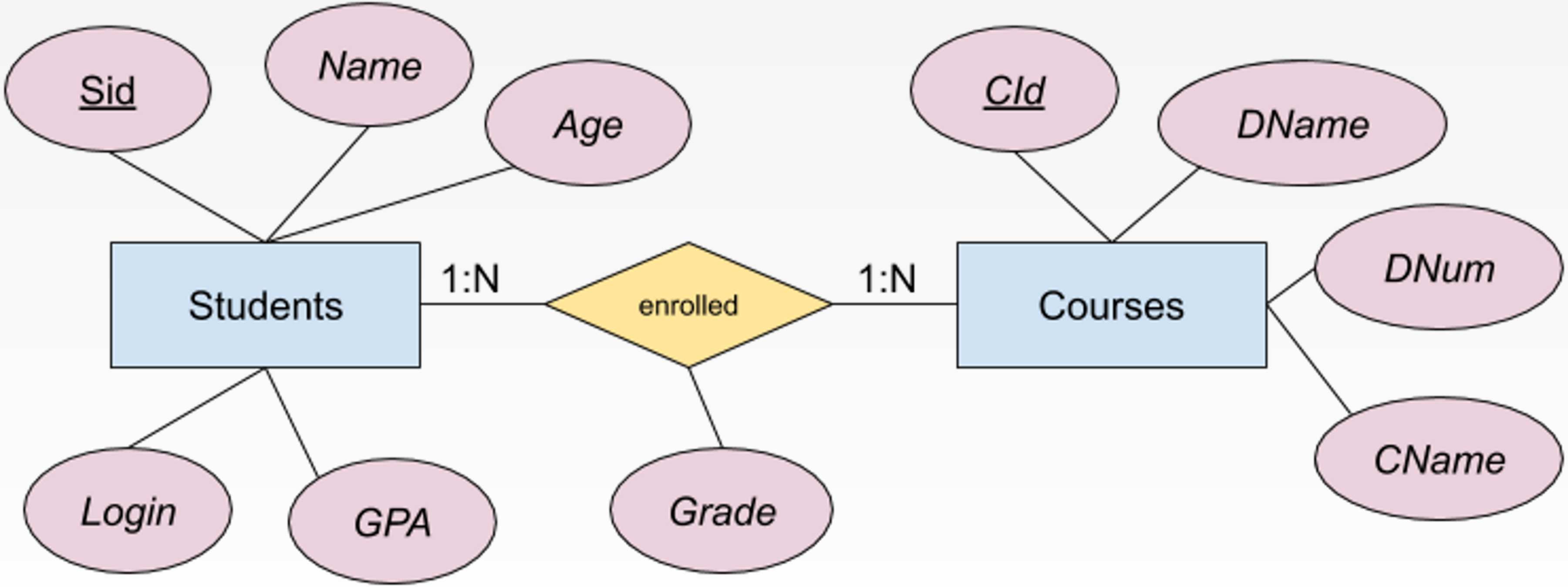
- **SQL Data Definition Language (DDL)**
  - Define and modify database schema
- **SQL Data Manipulation Language (DML)**
  - Manipulate data present in the database
  - Queries can be written intuitively
- Other Parts
  - SQL views
  - SQL indexes
  - SQL privileges

SQL DDL

- SQL CREATE statement
- SQL ALTER statement
- SQL DROP statement
- And more…

DDL Concepts

- **Usually,** a schema is a collection of tables and a Database is a collection of schemas
- SQL **schema** is a grouping of tables and other database objects such as views, constraints, and indexes which logically belong together
  CREATE SCHEMA PURCHASE **AUTHORIZATION** BBAESENS
- SQL table implements a relation from the relational model
  CREATE TABLE PRODUCT …
  CREATE TABLE PURCHASE.PRODUCT …


Sid
Name
Age
CId
DName
Students
1:N
enrolled
1:N
Courses
DNum
CName
Login
GPA
Grade

Creating Relations in SQL

- Creates the Students relation. Observe that the type (domain) of each field is specified, and enforced by the DBMS whenever tuples are added or modified.

```
CREATE TABLE Students
   (sid: CHAR(20),
    name: CHAR(20),
    login: CHAR(10),
    age: INTEGER,
    gpa: REAL)
```

Creating Relations in SQL

- Creates the Students relation. Observe that the type (domain) of each field is specified, and enforced by the DBMS whenever tuples are added or modified.
- As another example, the Enrolled table holds information about courses that students take.

```
CREATE TABLE Students
   (sid: CHAR(20),
    name: CHAR(20),
    login: CHAR(10),
    age: INTEGER,
    gpa: REAL)
CREATE TABLE Enrolled
   (sid: CHAR(20),
    cid: CHAR(20),
    grade: CHAR(2))
```

Relationship Sets to Tables

In translating a **many-to-many** relationship set to a relation, attributes of the relation must include:

1) Keys for each participating entity set (as foreign keys). This set of attributes forms a ***superkey*** for the relation.

2) All descriptive attributes.

```
CREATE TABLE Works_In(
  ssn  CHAR(1),
  did  INTEGER,
  since  DATE,
  PRIMARY KEY (ssn, did),
  FOREIGN KEY (ssn)
    REFERENCES Employees,
  FOREIGN KEY (did)
    REFERENCES Departments)
```

ssn	did	since
123-22-3666	51	1/1/91
123-22-3666	56	3/3/93
231-31-5368	51	2/2/92

Data Types

Data type	Description
`CHAR(n)`	Holds a fixed-length string with size *n*
`VARCHAR(n)`	Holds a variable-length string with maximum size *n*
`SMALLINT`	Small integer (no decimal) between -32768 and 32767
`INT`	Integer (no decimal) between -2147483648 and 2147483647
`FLOAT(n,d)`	Small number with a floating decimal point. The total maximum number of digits is *n* with a maximum of *d* digits to the right of the decimal point
`DOUBLE(n,d)`	Large number with a floating decimal point. The total maximum number of digits is *n* with a maximum of *d* digits to the right of the decimal point
`DATE`	Date in format YYYY-MM-DD
`DATETIME`	Date and time in format YYYY-MM-DD HH:MI:SS
`TIME`	Time in format HH:MI:SS
`BOOLEAN`	True or false
`BLOB`	Binary large object (e.g., image, audio, video)

User-defined Data Types

- **CREATE DOMAIN** creates a new domain. A domain is essentially a data type with optional constraints. The user who defines a domain becomes its owner.
- **CREATE DOMAIN PRODTYPE** AS VARCHAR(10)
  CHECK (VALUE IN ('white', 'red', 'rose', 'sparkling'))
- **CREATE DOMAIN CPI_DATA** AS INT CHECK (value >= 0 AND value <= 10);
- CREATE TABLE student(
  sid char(9) PRIMARY KEY,
  name varchar(30),
  cpi **CPI_DATA**);
- PostgreSQL (supported). MySQL (not supported)

Create Table Statement

```
CREATE TABLE SUPPLIER
(SUPNR CHAR(4) NOT NULL PRIMARY KEY,
 SUPNAME VARCHAR(40) NOT NULL,
 SUPADDRESS VARCHAR(50),
 SUPCITY VARCHAR(20),
 SUPSTATUS SMALLINT)

CREATE TABLE PRODUCT
(PRODNR CHAR(6) NOT NULL PRIMARY KEY,
 PRODNAME VARCHAR(60) NOT NULL,
 CONSTRAINT UC1 UNIQUE(PRODNAME),
 PRODTYPE VARCHAR(10),
 CONSTRAINT CC1 CHECK(PRODTYPE IN ('white', 'red', 'rose','sparkling')),
 AVAILABLE_QUANTITY INTEGER)
```

Create Table Statement

```
CREATE TABLE SUPPLIES
(SUPNR CHAR(4) NOT NULL,
 PRODNR CHAR(6) NOT NULL,
 PURCHASE_PRICE DOUBLE(8,2)
 COMMENT 'PURCHASE_PRICE IN EUR',
 DELIV_PERIOD TIME
 COMMENT 'DELIV_PERIOD IN DAYS',
 PRIMARY KEY (SUPNR, PRODNR),
 FOREIGN KEY (SUPNR) REFERENCES SUPPLIER (SUPNR)
 ON DELETE CASCADE ON UPDATE CASCADE,
 FOREIGN KEY (PRODNR) REFERENCES PRODUCT (PRODNR)
 ON DELETE CASCADE ON UPDATE CASCADE)
```

Create Table Statement

```
CREATE TABLE PURCHASE_ORDER
(PONR CHAR(7) NOT NULL PRIMARY KEY,
 PODATE DATE,
 SUPNR CHAR(4) NOT NULL,
 FOREIGN KEY (SUPNR) REFERENCES SUPPLIER (SUPNR)
 ON DELETE CASCADE ON UPDATE CASCADE)

CREATE TABLE PO_LINE
 (PONR CHAR(7) NOT NULL,
  PRODNR CHAR(6) NOT NULL,
  QUANTITY INTEGER,
  PRIMARY KEY (PONR, PRODNR),
  FOREIGN KEY (PONR) REFERENCES PURCHASE_ORDER (PONR)
  ON DELETE CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (PRODNR) REFERENCES PRODUCT (PRODNR)
  ON DELETE CASCADE ON UPDATE CASCADE)
```

DROP

- **DROP** command can be used to drop or remove database objects
  - Can also be combined with CASCADE and RESTRICT
  - Destroy relation: The schema information and the tuples are deleted.
- Examples:

DROP SCHEMA PURCHASE **CASCADE**

DROP SCHEMA PURCHASE **RESTRICT**

DROP TABLE PRODUCT

DROP TABLE PRODUCT **CASCADE**

DROP TABLE PRODUCT **RESTRICT**

Alter Relations

- **ALTER** statement can be used to modify table column definitions
- Examples:

  ALTER TABLE Students ADD COLUMN firstYear: integer

  The schema of Students is altered by adding a new field **firstYear**; every tuple in the current instance is extended with a null value in the new field

  ALTER TABLE PRODUCT **ADD** PRODIMAGE **BLOB**

  ALTER TABLE SUPPLIER **ALTER** SUPSTATUS **SET DEFAULT** '10'

Truncate

- Drop and re-create the table, which is much faster than deleting rows one by one, particularly for large tables.
- Cause an implicit commit, and so cannot be rolled back.
- **Truncate** table student
- vs DML – **Delete**
  - **Delete** from student

Integrity Constraints (ICs)

- IC: condition that **must** be true for *any* instance of the database; e.g., *domain constraints.*
  - ICs are specified when schema is defined (or altered).
  - ICs are checked when tuples are modified.
- A *legal* instance of a relation is one that satisfies all specified ICs.
  - DBMS should not allow illegal instances.
- If the DBMS checks ICs, stored data is more faithful to real-world meaning.
  - Avoids data entry errors, too!

Constraints

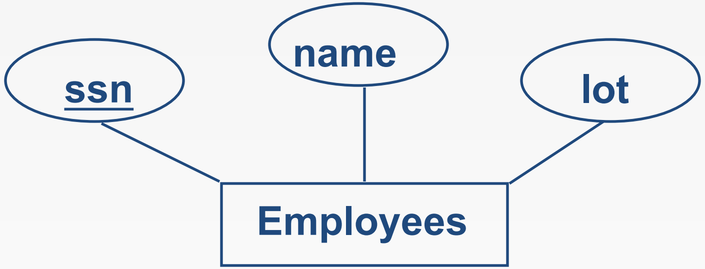
- **Column constraints**
  - **PRIMARY KEY** constraint defines the primary key of the table
  - **FOREIGN KEY** constraint defines a foreign key of a table
  - **UNIQUE** constraint defines an alternative key of a table
  - **NOT NULL** constraint prohibits null values for a column
  - **CHECK** constraint defines a constraint on the column values
  - **DEFAULT** constraint sets a default value for a column

Primary Key Constraints

- A set of fields is a *key* for a relation if :
  1. No two distinct tuples can have same values in all key fields, and
  2. This is not true for any subset of the key.
  - Part 2 false? A *superkey*.
  - If there's >1 key for a relation, one of the keys is chosen to be the *primary key*.
- E.g., *sid* is a key for Students.
  - The set {*sid, gpa*} is a superkey.

Primary Keys in SQL

- Entity sets to tables. Easy.



ssn	name	lot
123-22-3666	Attishoo	48
231-31-5368	Smiley	22
131-24-3650	Smethurst	35

```
CREATE TABLE Employees
 (ssn CHAR(11),
  name CHAR(20),
  lot  INTEGER,
  PRIMARY KEY  (ssn))
```

Primary and Candidate Keys in SQL

- Possibly many *candidate keys* (specified using UNIQUE), one of which is chosen as the *primary key*.

What is the difference between these two relations?

```
CREATE TABLE Enrolled
   (sid CHAR(20)
    cid  CHAR(20),
    grade CHAR(2),
    PRIMARY KEY(sid,cid))
CREATE TABLE Enrolled
   (sid CHAR(20)
    cid  CHAR(20),
    grade CHAR(2),
    PRIMARY KEY (sid),
    UNIQUE (cid, grade))
```

Primary and Candidate Keys in SQL

- Possibly many *candidate keys* (specified using UNIQUE), one of which is chosen as the *primary key*.

"For a given student and course, there is a single grade." vs. *"Students can take only one course, and receive a single grade for that course; further, no two students in a course receive the same grade."*

```
CREATE TABLE Enrolled
   (sid CHAR(20)
    cid  CHAR(20),
    grade CHAR(2),
    PRIMARY KEY(sid,cid))
```

```
CREATE TABLE Enrolled
   (sid CHAR(20)
    cid  CHAR(20),
    grade CHAR(2),
    PRIMARY KEY (sid),
    UNIQUE (cid, grade))
```

Used carelessly, an IC can prevent the storage of database instances that arise in practice!

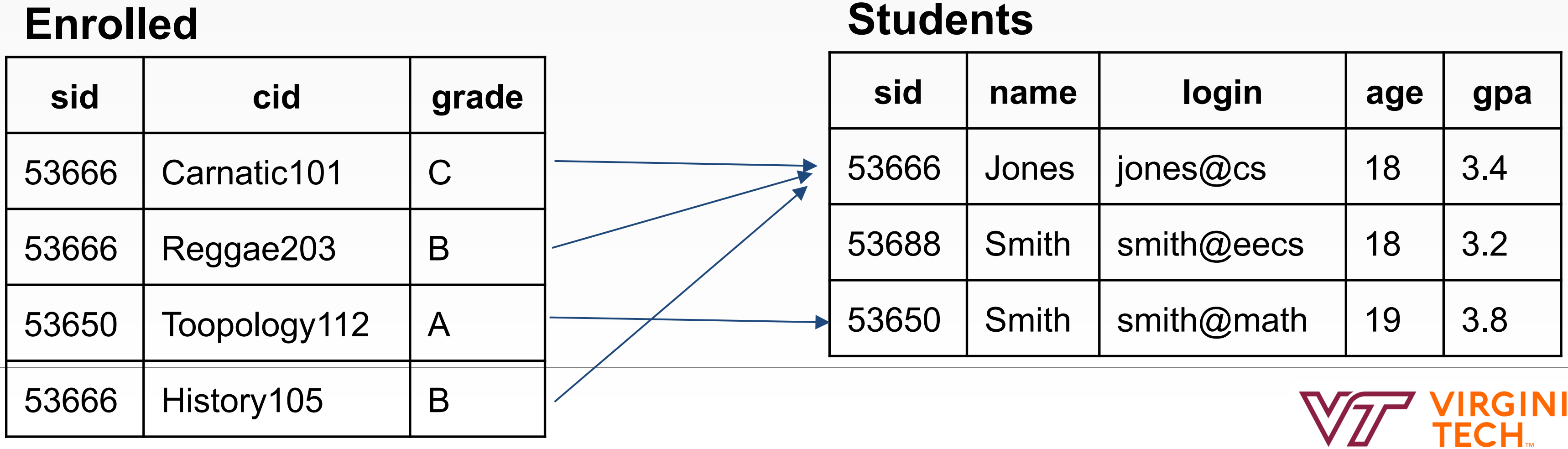
Foreign Keys, Referential Integrity

- *Foreign key* : Set of fields in one relation that is used to `refer’ to a tuple in another relation. (Must correspond to primary key of the second relation.) Like a `logical pointer’.
- E.g. *sid* is a foreign key referring to Students:
  - Enrolled(*sid*: string, *cid*: string, *grade*: string)
  - If all foreign key constraints are enforced, *referential integrity* is achieved, i.e., no dangling references.

Foreign Keys in SQL

- Only students listed in the Students relation should be allowed to enroll for courses.

```
CREATE TABLE Enrolled
   (sid CHAR(20),  cid CHAR(20),  grade CHAR(2),
     PRIMARY KEY  (sid,cid),
     FOREIGN KEY (sid) REFERENCES Students )
```



Enrolled

sid	cid	grade
53666	Carnatic101	C
53666	Reggae203	B
53650	Toopology112	A
53666	History105	B

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Enforcing Referential Integrity

- Consider Students and Enrolled; *sid* in Enrolled is a foreign key that references Students.
- What should be done if an Enrolled tuple with a non-existent student id is inserted? (*Reject it!*)

Enforcing Referential Integrity

- Consider Students and Enrolled; *sid* in Enrolled is a foreign key that references Students.
- What should be done if an Enrolled tuple with a non-existent student id is inserted? (*Reject it!*)
- What should be done if a Students tuple is deleted?
  - Also delete all Enrolled tuples that refer to it.
  - Disallow deletion of a Students tuple that is referred to.
  - Set sid in Enrolled tuples that refer to it to a *default sid*.
  - (In SQL, also: Set sid in Enrolled tuples that refer to it to a special value *null*, denoting *`unknown'* or *`inapplicable'*.)
- Similar if primary key of Students tuple is updated.

Referential Integrity Constraints

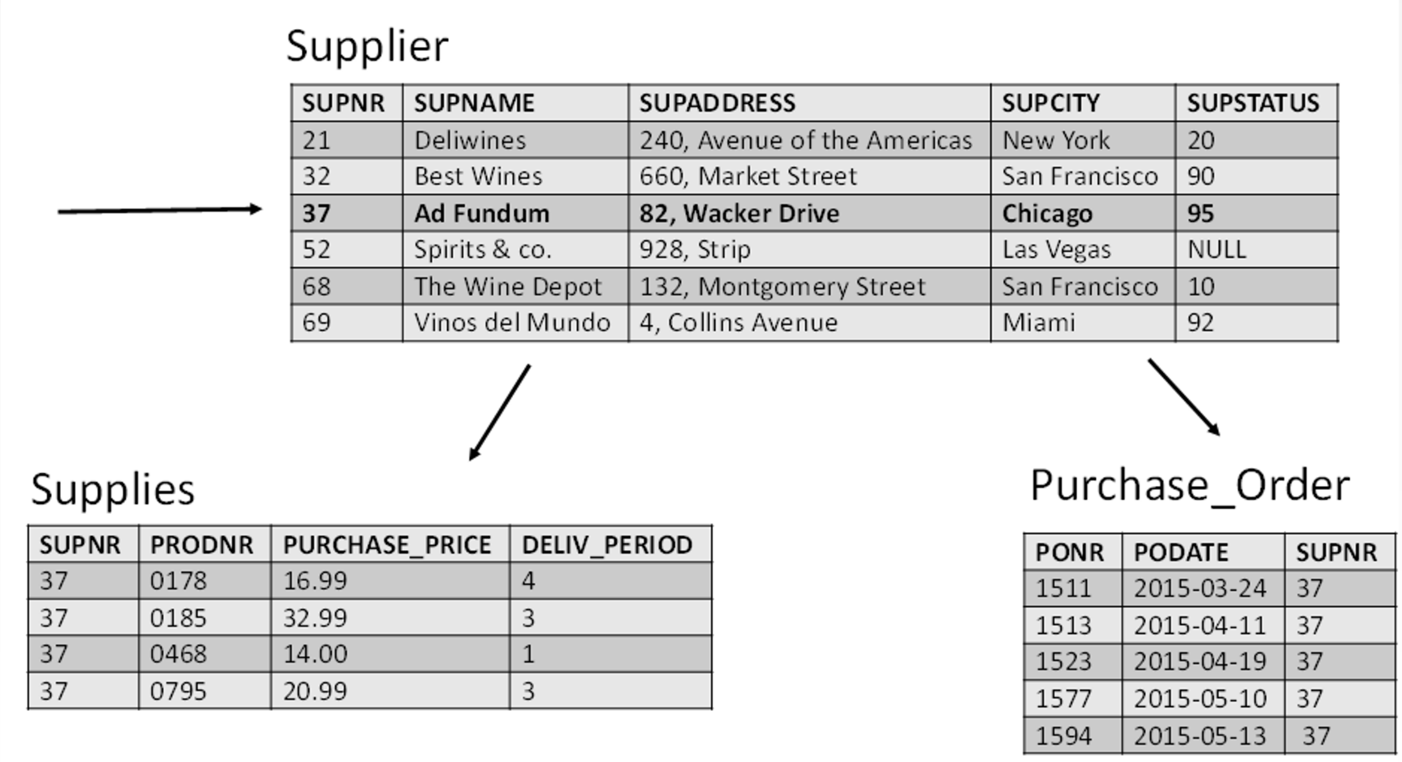
- What should happen to foreign keys in case a primary key is updated or deleted?
- SQL/92 and SQL:1999 support all 4 options on deletes and updates.
  - Default is NO ACTION (*delete/update is rejected*)
  - CASCADE (also delete all tuples that refer to deleted tuple)
  - SET NULL / SET DEFAULT (sets foreign key value of referencing tuple)

```
CREATE TABLE Enrolled
   (sid CHAR(20),
    cid CHAR(20),
    grade CHAR(2),
    PRIMARY KEY  (sid,cid),
    FOREIGN KEY (sid)
      REFERENCES Students
   ON DELETE CASCADE
   ON UPDATE SET NULL)
```

Referential Integrity Constraints

- Foreign key has the same domain as the primary key it refers to and either occurs as a value of it or NULL
- Options:
  - **ON UPDATE/DELETE RESTRICT:** update/removal is halted if referring tuples exist
  - **ON UPDATE/DELETE CASCADE**: update/removal should be cascaded to all referring tuples
  - **ON UPDATE/DELETE SET NULL:** foreign keys in the referring tuples are set to NULL
  - **ON UPDATE/DELETE SET DEFAULT:** foreign keys in the referring tuples are set to their default value

Referential Integrity Constraints



Supplier

SUPNR	SUPNAME	SUPADDRESS	SUPCITY	SUPSTATUS
21	Deliwines	240, Avenue of the Americas	New York	20
32	Best Wines	660, Market Street	San Francisco	90
37	**Ad Fundum**	**82, Wacker Drive**	**Chicago**	**95**
52	Spirits & co.	928, Strip	Las Vegas	NULL
68	The Wine Depot	132, Montgomery Street	San Francisco	10
69	Vinos del Mundo	4, Collins Avenue	Miami	92

Supplies

SUPNR	PRODNR	PURCHASE_PRICE	DELIV_PERIOD
37	0178	16.99	4
37	0185	32.99	3
37	0468	14.00	1
37	0795	20.99	3

Purchase_Order

PONR	PODATE	SUPNR
1511	2015-03-24	37
1513	2015-04-11	37
1523	2015-04-19	37
1577	2015-05-10	37
1594	2015-05-13	37

Check Constraints

- Allow you to make assertions about the data being inserted or updated.

```
CREATE TABLE Enrolled
   (sid CHAR(20),
    cid CHAR(20),
    grade CHAR(2),
    PRIMARY KEY  (sid,cid),
    FOREIGN KEY (sid)
      REFERENCES Students
   ON DELETE CASCADE
   ON UPDATE SET DEFAULT
    CHECK (grade in (‘A’, ‘B’,
‘C’, ‘D’, ‘F’) )
```

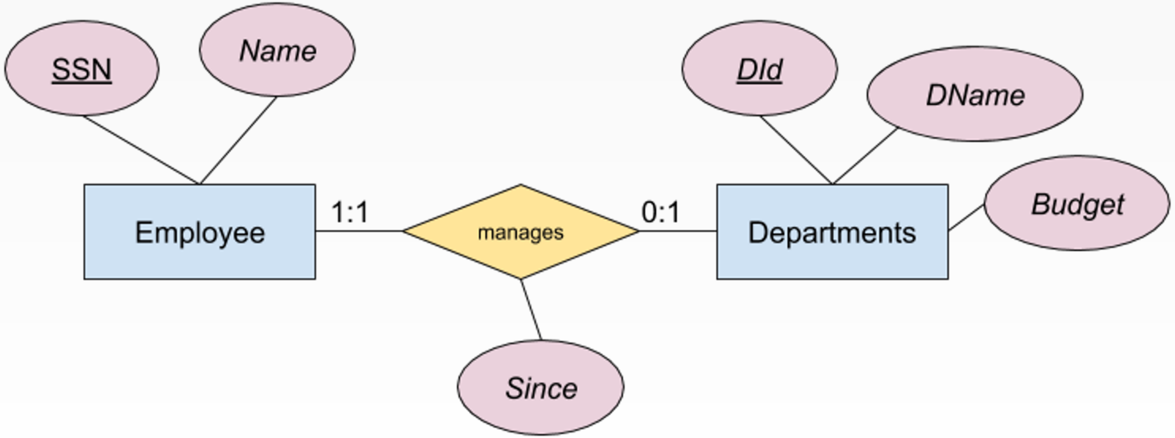
Check Constraints

- Can also compare to other columns in the tuple:

```
CREATE TABLE Ranges
   (min INT,
    max INT,
    PRIMARY KEY  (min, max),
    CHECK (min < max))
```

Review: Key + Participation Constraints

Every department has one manager.
Every did value in Departments table must appear in a row of the Manages table (with a non-null ssn value!)



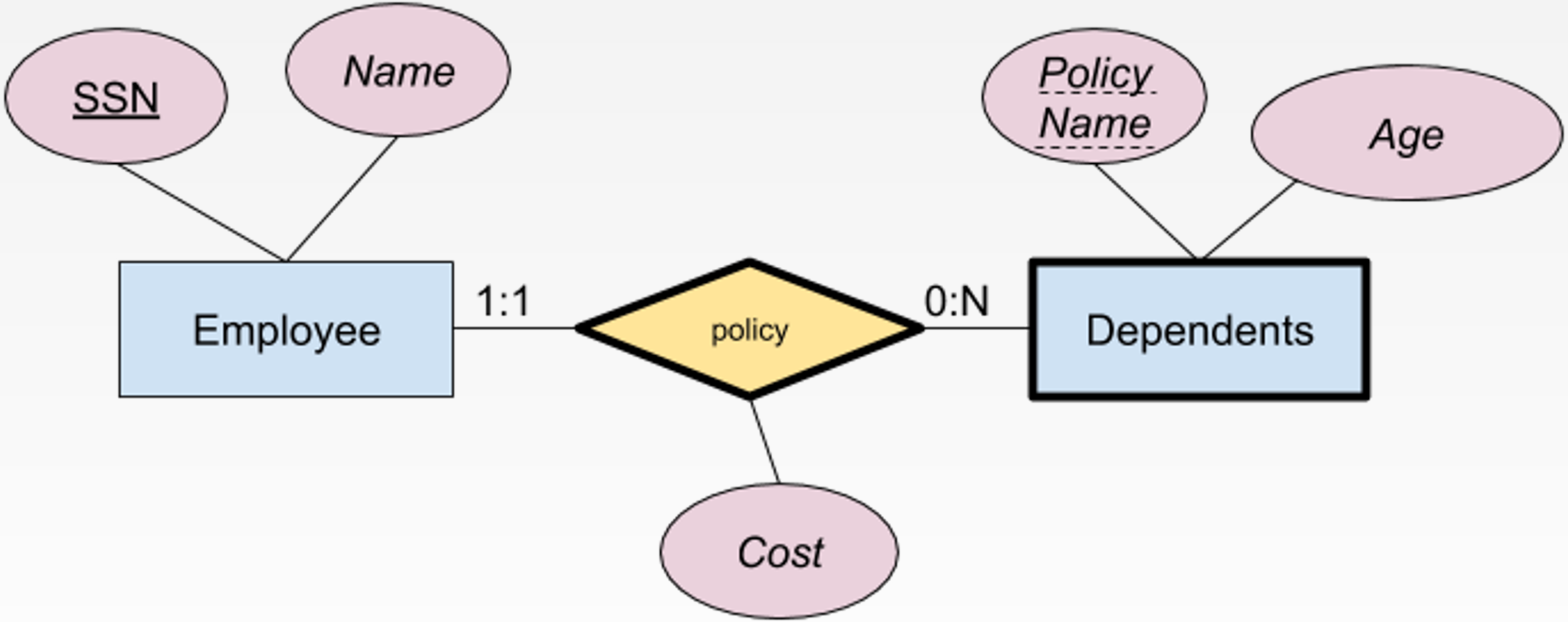
Participation Constraints in SQL

- We can capture participation constraints involving one entity set in a binary relationship, but little else.

```
CREATE TABLE  Dept_Mgr(
   did  INTEGER,
   dname  CHAR(20),
   budget  REAL,
   ssn  CHAR(11) NOT NULL,
   since  DATE,
   PRIMARY KEY  (did),
   FOREIGN KEY  (ssn) REFERENCES Employees,
      ON DELETE NO ACTION)
```

Review: Weak Entities

- A *weak entity* can be identified uniquely only by considering the primary key of another (*owner*) entity.
  - Owner entity set and weak entity set must participate in a one-to-many relationship set (1 owner, many weak entities).
  - Weak entity set must have total participation in this *identifying* relationship set.

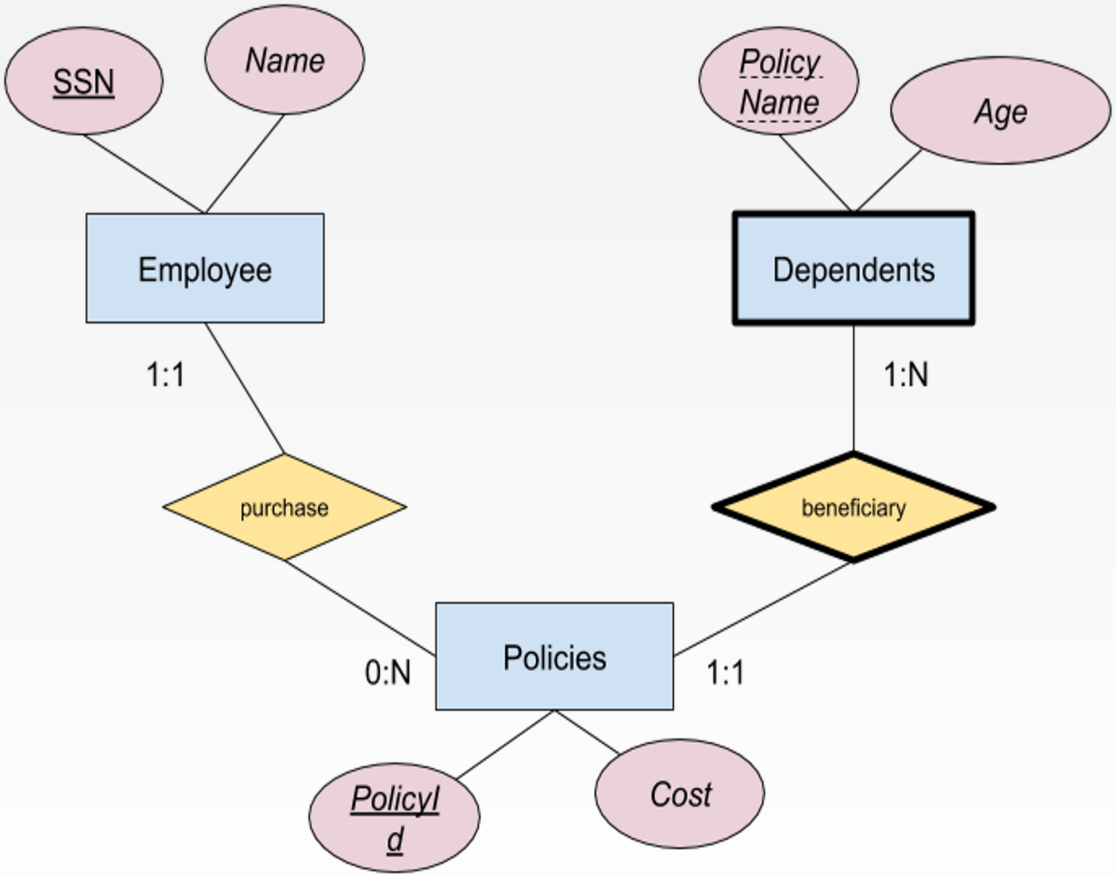

SSN
Name
Policy Name
Age
Employee
1:1
policy
0:N
Dependents
Cost

Translating Weak Entity Sets

- Weak entity set and identifying relationship set are translated into a single table.
  - When the owner entity is deleted, all owned weak entities must also be deleted.

```
CREATE TABLE  Dep_Policy (
   policy_name  CHAR(20),
   age  INTEGER,
   cost REAL,
   ssn  CHAR(11) NOT NULL,
   PRIMARY KEY  (policy_name, ssn),
   FOREIGN KEY  (ssn) REFERENCES Employees,
      ON DELETE CASCADE)
```

- Employees purchase policies. Every policy is purchased by exactly one employee (key – many-to-one + participation).
- Policies benefit dependents. Every dependent is covered by exactly one policy (key – many-to-one + participation). Dependents are uniquely identified by their pname and the policy covering them (weak).

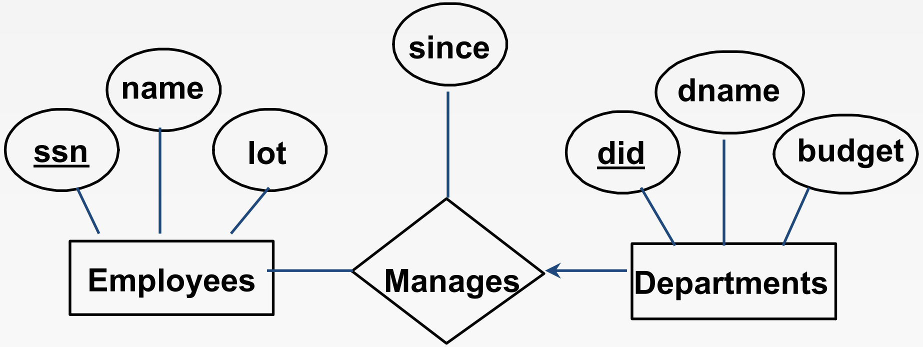


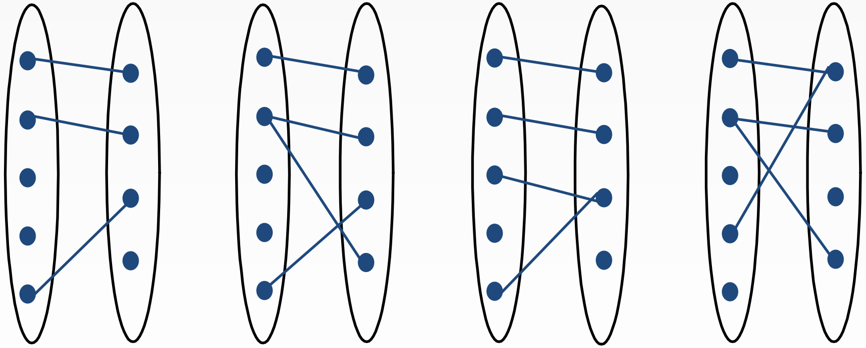
- The key constraints allow us to combine Purchaser with Policies and Beneficiary with Dependents.
- Participation constraints lead to NOT NULL constraints.

```
 CREATE TABLE  Policies (
     policyid  INTEGER NOT NULL,
     cost  REAL,
     ssn  CHAR(11)  NOT NULL,
     PRIMARY KEY (policyid).
     FOREIGN KEY (ssn) REFERENCES Employees,
          ON DELETE CASCADE)
CREATE TABLE Dependents (
    policy_name  CHAR(20) NOT NULL,
    age  INTEGER,
    policyid  INTEGER NOT NULL,
    ssn  CHAR(11)  NOT NULL,
    PRIMARY KEY (policy_name, policyid).
    FOREIGN KEY (policyid) REFERENCES Policies,
         ON DELETE CASCADE)
```

Review: Key Constraints

Each dept has at most one manager, according to the **key constraint** on Manages.





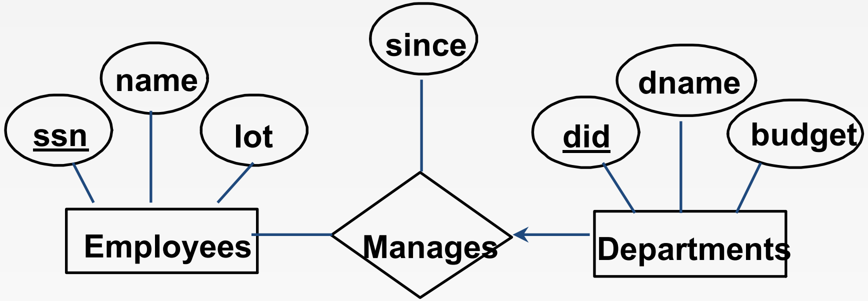
1-to-1

1-to Many

Many-to-1

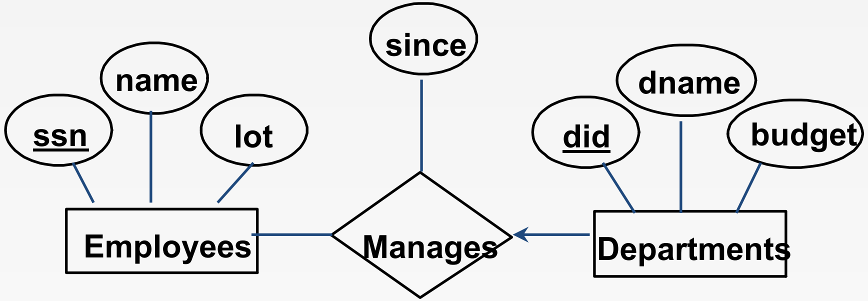
Many-to-Many

Translating ER with Key Constraints



```
CREATE TABLE  Manages(
 ssn  CHAR(11),
 did  INTEGER,
 since  DATE,
 PRIMARY KEY  (did),
 FOREIGN KEY (ssn)
   REFERENCES Employees,
 FOREIGN KEY (did)
   REFERENCES Departments)
```

Translating ER with Key Constraints, cont



Since each department has a unique manager, we could instead combine Manages and Departments.

```
CREATE TABLE  Manages(
 ssn  CHAR(11),
 did  INTEGER,
 since  DATE,
 PRIMARY KEY  (did),
 FOREIGN KEY (ssn)
   REFERENCES Employees,
 FOREIGN KEY (did)
   REFERENCES Departments)
```

Vs.

```
CREATE TABLE  Dept_Mgr(
 did  INTEGER,
 dname  CHAR(20),
 budget  REAL,
 ssn  CHAR(11),
 since  DATE,
 PRIMARY KEY  (did),
 FOREIGN KEY (ssn)
   REFERENCES Employees)
```

SQL DML

- SQL INSERT statement
- SQL DELETE statement
- SQL UPDATE statement
- SQL SELECT statement

SQL INSERT Statement

```
INSERT INTO PRODUCT VALUES
('980', 'Chateau Angelus, Grand Clu Classé, 1960', 'red',
6)
```

```
INSERT INTO PRODUCT(PRODNR, PRODNAME, PRODTYPE,
AVAILABLE_QUANTITY) VALUES
('980', 'Chateau Angelus, Grand Clu Classé, 1960', 'red',
6)
```

```
INSERT INTO PRODUCT(PRODNR, PRODNAME, PRODTYPE) VALUES
('980', 'Chateau Angelus, Grand Clu Classé, 1960', 'red')
```

SQL INSERT Statement

```
INSERT INTO PRODUCT(PRODNR, PRODNAME, PRODTYPE,
AVAILABLE_QUANTITY) VALUES
('980', 'Chateau Angelus, Grand Clu Classé, 1960', 'red',
6),
('1000', 'Domaine de la Vougeraie, Bâtard Montrachet',
Grand cru, 2010’, 'white', 2),
('1002', 'Leeuwin Estate Cabernet Sauvignon 2011',
'white', 20)

INSERT INTO INACTIVE-SUPPLIERS(SUPNR)
SELECT SUPNR FROM SUPPLIER
   EXCEPT
   SELECT SUPNR FROM SUPPLIES
```

SQL DELETE Statement

```
DELETE FROM PRODUCT
WHERE PRODNR = '1000'


DELETE FROM SUPPLIER
WHERE SUPSTATUS IS NULL


DELETE FROM SUPPLIES
WHERE PRODNR IN (SELECT PRODNR
           FROM PRODUCT
           WHERE PRODNAME LIKE '%CHARD%')
```

SQL DELETE Statement

```
DELETE FROM SUPPLIER R
WHERE NOT EXISTS
   (SELECT PRODNR
    FROM SUPPLIES S
    WHERE R.SUPNR = S.SUPNR)

DELETE FROM SUPPLIES S1
WHERE S1.PURCHASE_PRICE >
(SELECT 2 * AVG(S2.PURCHASE_PRICE)
FROM SUPPLIES S2
WHERE S1.PRODNR = S2.PRODNR)

DELETE FROM PRODUCT
```

SQL UPDATE Statement

```
UPDATE PRODUCT
SET AVAILABLE_QUANTITY = 26
WHERE PRODNR = '0185'


UPDATE SUPPLIER
SET SUPSTATUS = DEFAULT


UPDATE SUPPLIES
SET DELIV_PERIOD = DELIV_PERIOD+7
WHERE SUPNR IN (SELECT SUPNR
            FROM SUPPLIER
            WHERE SUPNAME = 'Deliwines')
```

SQL UPDATE Statement

```
UPDATE SUPPLIES S1
SET (PURCHASE_PRICE, DELIV_PERIOD) =
(SELECT MIN(PURCHASE_PRICE), MIN(DELIV_PERIOD)
FROM SUPPLIES S2
WHERE S1.PRODNR = S2.PRODNR)
WHERE SUPNR = '68'

ALTER TABLE SUPPLIER ADD SUPCATEGORY VARCHAR(10) DEFAULT
'SILVER'
UPDATE SUPPLIER SET SUPCATEGORY =
CASE WHEN SUPSTATUS >= 70 AND SUPSTATUS <= 90 THEN 'GOLD'
WHEN SUPSTATUS >= 90 THEN 'PLATINUM’ ELSE 'SILVER'
END
```

SELECT Statement

SELECT [DISTINCT] <column expression list>

FROM <single table>

[**WHERE** <predicate>]

[**ORDER BY** <column list>]

[**GROUP BY** <column list>]

[**HAVING** <predicate>]

[**LIMIT** <integer>]

SELECT Statement Overview

- The result of an SQL SELECT statement is a multiset, and not a set!
- In a multiset (aka bag), the elements are not ordered and there can be duplicates
- Examples: set {10, 5, 20} and multiset {10, 5, 10, 20, 5, 10}
- SQL does not eliminate duplicates
  - Duplicate elimination is expensive
  - User may want to see duplicate tuples
  - Duplicates may be considered by aggregate functions

Basic Single-Table Queries

- **SELECT** [**DISTINCT**] <column expression list>
  FROM <single table>
  [**WHERE** <predicate>]

- Simplest version is straightforward
  - Produce all tuples in the table that satisfy the predicate
  - Output the expressions in the SELECT list
  - Expression can be a column reference, or an arithmetic expression (e.g., *, /) over column refs

Example: SELECT Statement

- Find all 27-year-old sailors:

```
SELECT *
FROM  Sailors AS S
WHERE S.age=27;
```

- To find just names and rating, replace the first line to:

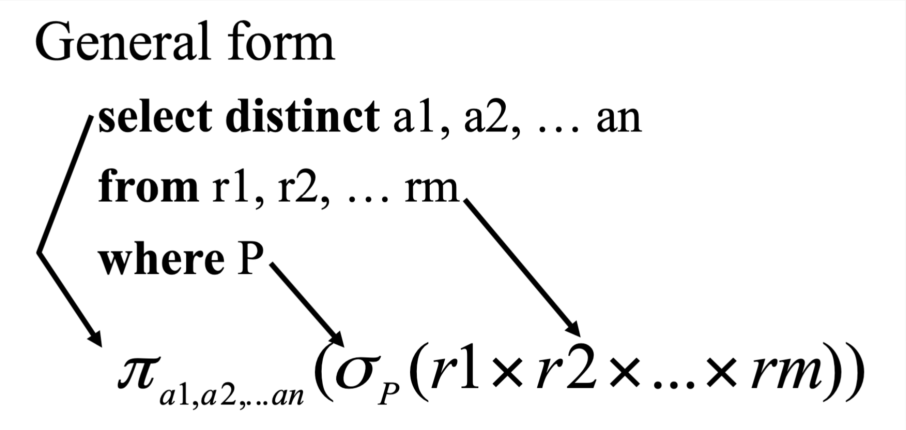
```
SELECT S.sname, S.rating
FROM  Sailors AS S
WHERE S.age=27 and rating > 5;
```

Sailors

sid	sname	rating	age
1	Fred	7	22
2	Jim	2	39
3	Nancy	8	27

SELECT Statement vs Relational Algebra

- Relational Algebra is set semantics (everything is a set), so removes duplicates automatically.
- SQL is bag semantics (everything is a multiset), so removes duplicates only when asked to (using **distinct**)



General form

select distinct a1, a2, … an

from r1, r2, … rm

where P

$$\pi_{a1,a2,\ldots an}(\sigma_P(r1 \times r2 \times \ldots \times rm))$$

Let’s Do Lab

- https://github.com/VTCourses/CS4604_Labs/
- 1.ddl_dml

SELECT DISTINCT

SELECT DISTINCT S.name, S.gpa
FROM students S

- DISTINCT specifies removal of duplicate rows before output
- Can refer to the students table as "S", this is called an alias

Simple Queries

SUPPLIER

SUPNR	SUPNAME	SUPADDRESS	SUPCITY	SUPSTATUS
21	Deliwines	240, Avenue of the Americas	New York	20
32	Best Wines	660, Market Street	San Francisco	90
37	Ad Fundum	82, Wacker Drive	Chicago	95
52	Spirits & co.	928, Strip	Las Vegas	NULL
68	The Wine Depot	132, Montgomery Street	San Francisco	10
69	Vinos del Mundo	4, Collins Avenue	Miami	92

Simple Queries

Q2: `SELECT SUPNR, SUPNAME FROM SUPPLIER`

SUPNR	SUPNAME
21	Deliwines
32	Best Wines
37	Ad Fundum
52	Spirits & co.
68	The Wine Depot
69	Vinos del Mundo

Simple Queries

Q3:

```
SELECT SUPNR
FROM PURCHASE_ORDER
```

SUPNR
32
32
37
37
37
37
37
68
69
94

Q4:

```
SELECT DISTINCT
SUPNR FROM
PURCHASE_ORDER
```

SUPNR
32
37
68
69
94

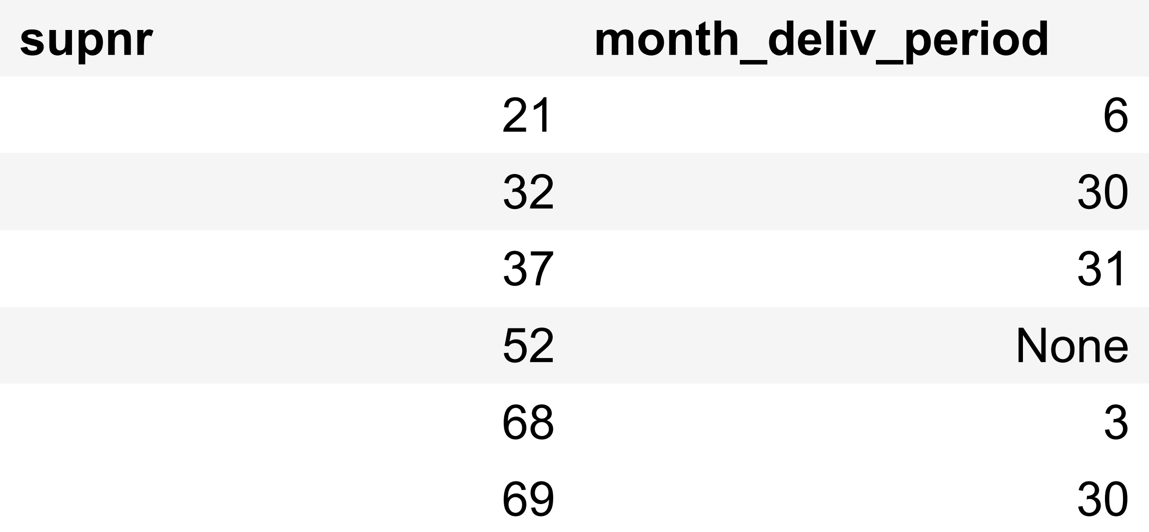
Renaming (Alias)

- **Rename a column use AS**
- **SELECT** column **AS** new_column_name **From** <single table>
- It is not the same as **RENAME COLUMN** or **CHANGE** in **DDL**

Simple Queries

Q5:

```
SELECT SUPNR, supstatus/3 AS
MONTH_DELIV_PERIOD FROM SUPPLIER
```



supnr	month_deliv_period
21	6
32	30
37	31
52	None
68	3
69	30

Where Clause

- Boolean operators (and or not ...)
- Comparison operators (<, >, =, ...)
- Wildcard Operators (%, _)
- Set-Comparison Operators (IN, NOT IN, EXISTS)
- and more...

Wildcard Operators

- find student ssns who live on “main” (st or str or street)

```
Select ssn
from student
where address like “main%”
```

- %: variable-length don’t care
- _: single-character don’t care

Simple Queries

SUPPLIER

SUPNR	SUPNAME	SUPADDRESS	SUPCITY	SUPSTATUS
21	Deliwines	240, Avenue of the Americas	New York	20
32	Best Wines	660, Market Street	San Francisco	90
37	Ad Fundum	82, Wacker Drive	Chicago	95
52	Spirits & co.	928, Strip	Las Vegas	NULL
68	The Wine Depot	132, Montgomery Street	San Francisco	10
69	Vinos del Mundo	4, Collins Avenue	Miami	92
94	The Wine Crate	182, Wacker Drive	Chicago	75

Simple Queries

Q6: **SELECT** SUPNR, SUPNAME **FROM** SUPPLIER
WHERE SUPCITY = 'San Francisco'

SUPNR	**SUPNAME**
32	Best Wines
68	The Wine Depot

Simple Queries

Q7:
```
SELECT SUPNR, SUPNAME  FROM SUPPLIER
WHERE SUPCITY = 'San Francisco' AND SUPSTATUS >
80
```

SUPNR	SUPNAME
32	Best Wines

Simple Queries

Q8:

```
SELECT SUPNR, SUPNAME, SUPSTATUS
FROM SUPPLIER WHERE SUPSTATUS BETWEEN 70 AND 80
```

SUPNR	SUPNAME	SUPSTATUS
94	The Wine Crate	75

Simple Queries

```
Q9: SELECT SUPNR, SUPNAME, SUPSTATUS
   FROM SUPPLIER WHERE SUPSTATUS IN (10, 90);
```

supnr	supname	supstatus
32	Best Wines	90
68	The Wine Depot	10

Simple Queries

Q10:
```
SELECT SUPNR, SUPNAME
FROM SUPPLIER
WHERE SUPNAME LIKE '%ine%'
```

supnr	supname
21	Deliwines
32	Best Wines
68	The Wine Depot
94	The Wine Crate

Note: underscore (_) is a substitute for a single character!

Simple Queries

```
Q11: SELECT SUPNR, SUPNAME
     FROM SUPPLIER
     WHERE SUPSTATUS IS NULL
```

SUPNR	SUPNAME
52	Spirits & Co.

Reading and Next Class

- SQL I: CH5
- Next: SQL II: CH5