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-one diagram](image)

one-many

![One-many diagram](image)
Recap: Cardinality

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

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

  ![Diagram](image1)

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

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

  ![Diagram](image2)
One to Many

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

![Diagram of participation constraints](image)
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
Executed by a program
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 …`
Creating Relations in SQL

- Creates the Students relation. Observe that the type \textit{(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.

```sql
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))
```
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.

```sql
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
)
```

<table>
<thead>
<tr>
<th>ssn</th>
<th>did</th>
<th>since</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-22-3666</td>
<td>51</td>
<td>1/1/91</td>
</tr>
<tr>
<td>123-22-3666</td>
<td>56</td>
<td>3/3/93</td>
</tr>
<tr>
<td>231-31-5368</td>
<td>51</td>
<td>2/2/92</td>
</tr>
</tbody>
</table>
### Data Types

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR(n)</td>
<td>Holds a fixed-length string with size n</td>
</tr>
<tr>
<td>VARCHAR(n)</td>
<td>Holds a variable-length string with maximum size n</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>Small integer (no decimal) between -32768 and 32767</td>
</tr>
<tr>
<td>INT</td>
<td>Integer (no decimal) between -2147483648 and 2147483647</td>
</tr>
<tr>
<td>FLOAT(n, d)</td>
<td>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</td>
</tr>
<tr>
<td>DOUBLE(n, d)</td>
<td>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</td>
</tr>
<tr>
<td>DATE</td>
<td>Date in format YYYY-MM-DD</td>
</tr>
<tr>
<td>DATETIME</td>
<td>Date and time in format YYYY-MM-DD HH:MI:SS</td>
</tr>
<tr>
<td>TIME</td>
<td>Time in format HH:MI:SS</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>True or false</td>
</tr>
<tr>
<td>BLOB</td>
<td>Binary large object (e.g., image, audio, video)</td>
</tr>
</tbody>
</table>
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 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:

  ```sql
  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.

  ```sql
  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 the 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.

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.

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

What is the difference between these two relations?

```
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.”

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

```
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))
```
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 )

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Carnatic101</td>
<td>C</td>
</tr>
<tr>
<td>53666</td>
<td>Reggae203</td>
<td>B</td>
</tr>
<tr>
<td>53650</td>
<td>Toopology112</td>
<td>A</td>
</tr>
<tr>
<td>53666</td>
<td>History105</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@eecs</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
</tr>
</tbody>
</table>
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

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

<table>
<thead>
<tr>
<th>Supplies</th>
<th>SUPNR</th>
<th>PRODNR</th>
<th>PURCHASE_PRICE</th>
<th>DELIV_PERIOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>0178</td>
<td>16.99</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>0185</td>
<td>32.99</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>0468</td>
<td>14.00</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>0795</td>
<td>20.99</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Purchase_Order</th>
<th>PONR</th>
<th>PO_DATE</th>
<th>SUPNR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1511</td>
<td>2015-03-24</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>1513</td>
<td>2015-04-11</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>1523</td>
<td>2015-04-19</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>1577</td>
<td>2015-05-10</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>1594</td>
<td>2015-05-13</td>
<td>37</td>
<td></td>
</tr>
</tbody>
</table>
Check Constraints

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

```sql
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:

```sql
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.

```sql
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.
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.

```sql
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.

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

```sql
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 INACTIVE-SUPPLIERS(SUPNR) SELECT SUPNR FROM SUPPLIER EXCEPT SELECT SUPNR FROM SUPPLIES
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%')
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
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**

```sql
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>]

VT
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

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Fred</td>
<td>7</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>Jim</td>
<td>2</td>
<td>39</td>
</tr>
<tr>
<td>3</td>
<td>Nancy</td>
<td>8</td>
<td>27</td>
</tr>
</tbody>
</table>
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

$$\pi_{a_1, a_2, \ldots, a_n}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$$
Let’s Do Lab

• [https://github.com/VTCourses/CS4604_Labs/](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

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

**Q2: SELECT SUPNR, SUPNAME FROM SUPPLIER**

<table>
<thead>
<tr>
<th>SUPNR</th>
<th>SUPNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>Deliwines</td>
</tr>
<tr>
<td>32</td>
<td>Best Wines</td>
</tr>
<tr>
<td>37</td>
<td>Ad Fundum</td>
</tr>
<tr>
<td>52</td>
<td>Spirits &amp; co.</td>
</tr>
<tr>
<td>68</td>
<td>The Wine Depot</td>
</tr>
<tr>
<td>69</td>
<td>Vinos del Mundo</td>
</tr>
</tbody>
</table>
Simple Queries

Q3: SELECT SUPNR
    FROM PURCHASE_ORDER

Q4: SELECT DISTINCT SUPNR
    FROM PURCHASE_ORDER
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`

<table>
<thead>
<tr>
<th>supnr</th>
<th>month_deliv_period</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>6</td>
</tr>
<tr>
<td>32</td>
<td>30</td>
</tr>
<tr>
<td>37</td>
<td>31</td>
</tr>
<tr>
<td>52</td>
<td>None</td>
</tr>
<tr>
<td>68</td>
<td>3</td>
</tr>
<tr>
<td>69</td>
<td>30</td>
</tr>
</tbody>
</table>
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

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

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

<table>
<thead>
<tr>
<th>SUPNR</th>
<th>SUPNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>Best Wines</td>
</tr>
<tr>
<td>68</td>
<td>The Wine Depot</td>
</tr>
</tbody>
</table>
Simple Queries

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

<table>
<thead>
<tr>
<th>SUPNR</th>
<th>SUPNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>Best Wines</td>
</tr>
</tbody>
</table>
Simple Queries

Q8: SELECT SUPNR, SUPNAME, SUPSTATUS
    FROM SUPPLIER WHERE SUPSTATUS BETWEEN 70 AND 80

<table>
<thead>
<tr>
<th>SUPNR</th>
<th>SUPNAME</th>
<th>SUPSTATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>94</td>
<td>The Wine Crate</td>
<td>75</td>
</tr>
</tbody>
</table>
Simple Queries

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

<table>
<thead>
<tr>
<th>supnr</th>
<th>supname</th>
<th>supstatus</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>Best Wines</td>
<td>90</td>
</tr>
<tr>
<td>68</td>
<td>The Wine Depot</td>
<td>10</td>
</tr>
</tbody>
</table>
Simple Queries

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

<table>
<thead>
<tr>
<th>supnr</th>
<th>supname</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>Deliwines</td>
</tr>
<tr>
<td>32</td>
<td>Best Wines</td>
</tr>
<tr>
<td>68</td>
<td>The Wine Depot</td>
</tr>
<tr>
<td>94</td>
<td>The Wine Crate</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>SUPNR</th>
<th>SUPNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>52</td>
<td>Spirits &amp; Co.</td>
</tr>
</tbody>
</table>
Reading and Next Class

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