BASIC SQL

CHAPTER 4 (6/E)
CHAPTER 8 (5/E)
LECTURE OUTLINE

- SQL Data Definition and Data Types
- Specifying Constraints in SQL
- Basic Retrieval Queries in SQL
- Set Operations in SQL
Structured Query Language

Considered one of the major reasons for the commercial success of relational databases

Statements for data definitions, queries, and updates

• Both DDL and DML
• Core specification plus specialized extensions

Terminology:

<table>
<thead>
<tr>
<th>Relational Model</th>
<th>SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>relation</td>
<td>table</td>
</tr>
<tr>
<td>tuple</td>
<td>row</td>
</tr>
<tr>
<td>attribute</td>
<td>column</td>
</tr>
</tbody>
</table>

Syntax notes:

• Some interfaces require each statement to end with a semicolon.
• SQL is not case-sensitive.
SQL DATA DEFINITION

- CREATE statement
  - Main SQL command for data definition

- SQL schema
  - Identified by a schema name
  - Includes an authorization identifier (owner)
  - Components are descriptors for each schema element
    - Tables, constraints, views, domains, and other constructs

- CREATE SCHEMA statement
  - CREATE SCHEMA COMPANY AUTHORIZATION 'Jsmith';
CREATE TABLE COMMAND

- Specify a new relation
  - Provide name
  - Specify attributes and initial constraints
- **Base tables (base relations)**
  - Relation and its tuples are physically stored and managed by DBMS
- Can optionally specify schema:
  - `CREATE TABLE COMPANY.EMPLOYEE ...`
  - or
  - `CREATE TABLE EMPLOYEE ...`
- Include information for each column (attribute) plus constraints
  - Column name
  - Column type (domain)
  - Key, uniqueness, and null constraints
BASIC DATA TYPES

- **Numeric** data types
  - Integer numbers: `INT`, `INTEGER`, `SMALLINT`, `BIGINT`
  - Floating-point (real) numbers: `REAL`, `DOUBLE`, `FLOAT`
  - Fixed-point numbers: `DECIMAL(n,m)`, `DEC(n,m)`, `NUMERIC(n,m)`, `NUM(n,m)`

- **Character-string** data types
  - Fixed length: `CHAR(n)`, `CHARACTER(n)`
  - Varying length: `VARCHAR(n)`, `CHAR VARYING(n)`, `CHARACTER VARYING(n)`, `LONG VARCHAR`

- **Large object** data types
  - Characters: `CLOB`, `CHAR LARGE OBJECT`, `CHARACTER LARGE OBJECT`
  - Bits: `BLOB`, `BINARY LARGE OBJECT`

- **Boolean** data type
  - Values of `TRUE` or `FALSE` or `NULL`

- **DATE** data type
  - Ten positions
  - Components are `YEAR`, `MONTH`, and `DAY` in the form `YYYY-MM-DD`
MORE DATA TYPES

- Additional data types
  - **TIMESTAMP** data type
    - Includes the DATE and TIME fields
    - Plus a minimum of six positions for decimal fractions of seconds
    - Optional WITH TIME ZONE qualifier
  - **INTERVAL** data type
    - Specifies a relative value that can be used to increment or decrement an absolute value of a date, time, or timestamp

- Columns can be declared to be **NOT NULL**
- Columns can be declared to have a default value
  - Assigned to column in any tuple for which a value is not specified
- Example
  ```sql
  CREATE TABLE EMPLOYEE (  
    ...  
    NICKNAME VARCHAR(20) DEFAULT NULL,  
    ...  
    Province CHAR(2) NOT NULL DEFAULT 'ON',  
    ...  
  );
  ```
**CREATE TABLE** EMPLOYEE

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fname</td>
<td>VARCHAR(15)</td>
<td>NOT NULL,</td>
</tr>
<tr>
<td>Minit</td>
<td>CHAR,</td>
<td>NOT NULL,</td>
</tr>
<tr>
<td>Lname</td>
<td>VARCHAR(15)</td>
<td>NOT NULL,</td>
</tr>
<tr>
<td>Ssn</td>
<td>CHAR(9)</td>
<td>NOT NULL,</td>
</tr>
<tr>
<td>Bdate</td>
<td>DATE,</td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>VARCHAR(30),</td>
<td></td>
</tr>
<tr>
<td>Sex</td>
<td>CHAR,</td>
<td></td>
</tr>
<tr>
<td>Salary</td>
<td>DECIMAL(10,2),</td>
<td></td>
</tr>
<tr>
<td>Super_ssn</td>
<td>CHAR(9),</td>
<td>NOT NULL )</td>
</tr>
<tr>
<td>Dno</td>
<td>INT</td>
<td></td>
</tr>
</tbody>
</table>

**CREATE TABLE** DEPARTMENT

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dname</td>
<td>VARCHAR(15)</td>
<td>NOT NULL,</td>
</tr>
<tr>
<td>Dnumber</td>
<td>INT</td>
<td>NOT NULL,</td>
</tr>
<tr>
<td>Mgr_ssn</td>
<td>CHAR(9)</td>
<td>NOT NULL,</td>
</tr>
<tr>
<td>Mgr_start_date</td>
<td>DATE</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 4.1**

SQL CREATE TABLE
DOMAINS IN SQL

- Name used in place of built-in data type
- Makes it easier to change the data type used by numerous columns
- Improves schema readability
- Example:
  
  ```sql
  CREATE DOMAIN SIN_TYPE AS CHAR(9);
  ```
SPECIFYING KEY CONSTRAINTS

- **PRIMARY KEY** clause
  - Specifies one or more attributes that make up the primary key of a relation
    
    Dnumber INT NOT NULL PRIMARY KEY,
  - Primary key attributes must be declared **NOT NULL**

- **UNIQUE** clause
  - Specifies alternate (candidate) keys
    
    Dname VARCHAR(15) UNIQUE;
  - May or may not allow null values, depending on declaration

- If no key constraints, two or more tuples may be identical in all columns.
  - SQL deviates from pure relational model!
  - Multiset (bag) behaviour
REFERENTIAL CONSTRAINTS

- **FOREIGN KEY clause**
  
  FOREIGN KEY (Dept) REFERENCES DEPARTMENT (Dnum),
  
  • Default operation: reject update on violation
  • Attach *referential triggered action* clause in case referenced tuple is deleted
    • Options include SET NULL, CASCADE, and SET DEFAULT

- Foreign key declaration must refer to a table already created
SPECIFYING TUPLE CONSTRAINTS

- Some constraints involve several columns
- **CHECK** clause at the end of a **CREATE TABLE** statement
  - Apply to each tuple individually
- Example
  - **CHECK** (Dept_create_date <= Mgr_start_date)
Recall Employee example:

**Example**

- **Employee**
  - Fname
  - Minit
  - Lname
  - Ssn
  - Bdate
  - Address
  - Sex
  - Salary
  - Super_ssn
  - Dno

- **Department**
  - Dname
  - Dnumber
  - Mgr_ssn
  - Mgr_start_date

- **Dept_locations**
  - Dnumber
  - Dlocation

- **Project**
  - Pname
  - Pnumber
  - Plocation
  - Dnum

- **Works_on**
  - Essn
  - Pno
  - Hours

- **Dependent**
  - Essn
  - Dependent_name
  - Sex
  - Bdate
  - Relationship

**Figure 3.7**
Referential integrity constraints displayed on the COMPANY relational database schema.
CREATE TABLE EMPLOYEE
(
  Dno INT NOT NULL DEFAULT 1,
CONSTRANT EMPPK
  PRIMARY KEY (Ssn),
CONSTRANT EMPSUPERFK
  FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn)
    ON DELETE SET NULL ON UPDATE CASCADE,
CONSTRANT EMPDEPTFK
  FOREIGN KEY(Dno) REFERENCES DEPARTMENT(Dnumber)
    ON DELETE SET DEFAULT ON UPDATE CASCADE);
CREATE TABLE DEPARTMENT
(
  Mgr_ssn CHAR(9) NOT NULL DEFAULT '888665555',
  . . .
CONSTRANT DEPTPK
  PRIMARY KEY(Dnumber),
CONSTRANT DEPTSK
  UNIQUE (Dname),
CONSTRANT DEPTMGRFK
  FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn)
    ON DELETE SET DEFAULT ON UPDATE CASCADE);
CREATE TABLE DEPT_LOCATIONS
(
  PRIMARY KEY (Dnumber, Dlocation),
CONSTRANT DEPTPK (Dnumber),
  FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber)
    ON DELETE CASCADE ON UPDATE CASCADE);

Figure 4.2
Example illustrating how default attribute values and referential integrity triggered actions are specified in SQL.
CREATE TABLE PROJECT
  ( Pname VARCHAR(15) NOT NULL,
    Pnumber INT NOT NULL,
    Plocation VARCHAR(15),
    Dnum INT NOT NULL,
  PRIMARY KEY (Pnumber),
  UNIQUE (Pname),
  FOREIGN KEY (Dnum) REFERENCES DEPARTMENT(Dnumber) );

CREATE TABLE WORKS_ON
  ( Essn CHAR(9) NOT NULL,
    Pno INT NOT NULL,
    Hours DECIMAL(3,1) NOT NULL,
  PRIMARY KEY (Essn, Pno),
  FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn),
  FOREIGN KEY (Pno) REFERENCES PROJECT(Pnumber) );

CREATE TABLE DEPENDENT
  ( Essn CHAR(9) NOT NULL,
    Dependent_name VARCHAR(15) NOT NULL,
    Sex CHAR,
    Bdate DATE,
    Relationship VARCHAR(8),
  PRIMARY KEY (Essn, Dependent_name),
  FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn) );
BASIC SQL RETRIEVAL QUERIES

- All retrievals use **SELECT** statement:

\[
\text{SELECT } \text{<return list>}
\text{ FROM } \text{<table list>}
\text{ [ WHERE } \text{<condition> } ]
\;
\]

where
- **<return list>** is a list of expressions or column names whose values are to be retrieved by the query
- **<table list>** is a list of relation names required to process the query
- **<condition>** is a Boolean expression that identifies the tuples to be retrieved by the query

- **Example**

\[
\begin{align*}
\text{SELECT } & \text{title, year, genre} \\
\text{FROM } & \text{Film} \\
\text{WHERE } & \text{director} = 'Steven Spielberg' \text{ AND year > 1990;}
\end{align*}
\]

- Omitting **WHERE** clause implies all tuples selected.
SEMANTICS FOR 1 RELATION

1. Start with the relation named in the FROM clause

2. Consider each tuple one after the other, eliminating those that do not satisfy the WHERE clause.
   - Boolean condition that must be true for any retrieved tuple
   - Logical comparison operators
     =, <, <=, >, >=, and <>

3. For each remaining tuple, create a return tuple with columns for each expression (column name) in the SELECT clause.
   - Use SELECT * to select all columns.

<table>
<thead>
<tr>
<th>Film</th>
<th>title</th>
<th>genre</th>
<th>year</th>
<th>director</th>
<th>minutes</th>
<th>budget</th>
<th>gross</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Company Men</td>
<td>The Company Men</td>
<td>drama</td>
<td>2010</td>
<td>John Wells</td>
<td>104</td>
<td>15,000,000</td>
<td>4,439,063</td>
</tr>
<tr>
<td>Lincoln</td>
<td>Lincoln</td>
<td>biography</td>
<td>2012</td>
<td>Steven Spielberg</td>
<td>150</td>
<td>65,000,000</td>
<td>181,408,467</td>
</tr>
<tr>
<td>War Horse</td>
<td>War Horse</td>
<td>drama</td>
<td>2011</td>
<td>Steven Spielberg</td>
<td>146</td>
<td>66,000,000</td>
<td>79,883,359</td>
</tr>
<tr>
<td>Argo</td>
<td>Argo</td>
<td>drama</td>
<td>2012</td>
<td>Ben Affleck</td>
<td>120</td>
<td>44,300,000</td>
<td>133,178,251</td>
</tr>
</tbody>
</table>
SELECT-FROM-WHERE SEMANTICS

- What if there are several relations in the FROM clause?

1. Start with cross-product of all relation(s) listed in the FROM clause.
   - Every tuple in $R_1$ paired up with every tuple in $R_2$ paired up with ...
2. Consider each tuple one after the other, eliminating those that do not satisfy the WHERE clause.
3. For each remaining tuple, create a return tuple with columns for each expression (column name) in the SELECT clause.

   *Steps 2 and 3 are just the same as before.*

```sql
SELECT actor, birth, movie
FROM Role, Person
WHERE actor = name and birth > 1940;
```

<table>
<thead>
<tr>
<th>Role</th>
<th>movie</th>
<th>persona</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ben Affleck</td>
<td>Argo</td>
<td>Tony Mendez</td>
</tr>
<tr>
<td>Alan Arkin</td>
<td>Argo</td>
<td>Lester Siegel</td>
</tr>
<tr>
<td>Ben Affleck</td>
<td>The Company Men</td>
<td>Bobby Walker</td>
</tr>
<tr>
<td>Tommy Lee Jones</td>
<td>The Company Men</td>
<td>Gene McClary</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Person</th>
<th>birth</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ben Affleck</td>
<td>1972</td>
<td>Berkeley</td>
</tr>
<tr>
<td>Alan Arkin</td>
<td>1934</td>
<td>New York</td>
</tr>
<tr>
<td>Tommy Lee Jones</td>
<td>1946</td>
<td>San Saba</td>
</tr>
</tbody>
</table>
AMBIGUOUS COLUMN NAMES

- Same name may be used for two (or more) columns (in different relations)
  - Must **qualify** the column name with the relation name to prevent ambiguity

<table>
<thead>
<tr>
<th>Customer</th>
<th>Sale</th>
<th>LineItem</th>
</tr>
</thead>
<tbody>
<tr>
<td>custid</td>
<td>saleid</td>
<td>saleid</td>
</tr>
<tr>
<td>name</td>
<td>date</td>
<td>product</td>
</tr>
<tr>
<td>address</td>
<td>custid</td>
<td>quantity</td>
</tr>
<tr>
<td>phone</td>
<td></td>
<td>price</td>
</tr>
</tbody>
</table>

```sql
SELECT name, date, product, quantity
FROM Customer, Sale, LineItem
WHERE price > 100 AND Customer.custid = Sale.custid AND
     Sale.saleid = LineItem.saleid;
```

- **Note**
  - If SELECT clause includes custid, it must specify whether to use Customer.custid or Sale.custid even though the values are guaranteed to be identical.
SELECT award, actor, persona, Role.movie
FROM Honours, Role
WHERE category = 'actor' AND winner = actor
    AND Honours.movie = Role.movie
### Figure 3.6
One possible database state for the COMPANY relational database schema.

#### EMPLOYEE

<table>
<thead>
<tr>
<th>Fname</th>
<th>Minit</th>
<th>Lname</th>
<th>Ssn</th>
<th>Bdate</th>
<th>Address</th>
<th>Sex</th>
<th>Salary</th>
<th>Super_ssn</th>
<th>Dno</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>B</td>
<td>Smith</td>
<td>123456789</td>
<td>1965-01-09</td>
<td>731 Fondren, Houston, TX</td>
<td>M</td>
<td>30000</td>
<td>333445555</td>
<td>5</td>
</tr>
<tr>
<td>Franklin</td>
<td>T</td>
<td>Wong</td>
<td>333445555</td>
<td>1955-12-08</td>
<td>638 Voss, Houston, TX</td>
<td>M</td>
<td>40000</td>
<td>888665555</td>
<td>5</td>
</tr>
<tr>
<td>Alicia</td>
<td>J</td>
<td>Zelaya</td>
<td>999887777</td>
<td>1968-01-19</td>
<td>3321 Castle, Spring, TX</td>
<td>F</td>
<td>25000</td>
<td>987654321</td>
<td>4</td>
</tr>
<tr>
<td>Jennifer</td>
<td>S</td>
<td>Wallace</td>
<td>987654321</td>
<td>1941-06-20</td>
<td>291 berry, Bellaire, TX</td>
<td>F</td>
<td>43000</td>
<td>888665555</td>
<td>4</td>
</tr>
<tr>
<td>Ramesh</td>
<td>K</td>
<td>Narayan</td>
<td>666884444</td>
<td>1962-09-15</td>
<td>975 fire Oak, Humble, TX</td>
<td>M</td>
<td>38000</td>
<td>333445555</td>
<td>5</td>
</tr>
<tr>
<td>Joyce</td>
<td>A</td>
<td>English</td>
<td>453453453</td>
<td>1972-07-31</td>
<td>5631 rice, Houston, TX</td>
<td>F</td>
<td>25000</td>
<td>333445555</td>
<td>5</td>
</tr>
<tr>
<td>Ahmad</td>
<td>V</td>
<td>Jabbar</td>
<td>987987987</td>
<td>1969-03-29</td>
<td>980 Dallas, Houston, TX</td>
<td>M</td>
<td>25000</td>
<td>987654321</td>
<td>4</td>
</tr>
<tr>
<td>James</td>
<td>E</td>
<td>Borg</td>
<td>888665555</td>
<td>1937-11-10</td>
<td>450 Stone, Houston, TX</td>
<td>M</td>
<td>55000</td>
<td>NULL</td>
<td>1</td>
</tr>
</tbody>
</table>

#### DEPARTMENT

<table>
<thead>
<tr>
<th>Dname</th>
<th>Dnumber</th>
<th>Mgr_ssn</th>
<th>Mgr_start_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research</td>
<td>5</td>
<td>333445555</td>
<td>1988-05-22</td>
</tr>
<tr>
<td>Administration</td>
<td>4</td>
<td>987654321</td>
<td>1995-01-01</td>
</tr>
<tr>
<td>Headquarters</td>
<td>1</td>
<td>888665555</td>
<td>1981-06-19</td>
</tr>
</tbody>
</table>

#### DEPT LOCATIONS

<table>
<thead>
<tr>
<th>Dnumber</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Houston</td>
</tr>
<tr>
<td>4</td>
<td>Stafford</td>
</tr>
<tr>
<td>5</td>
<td>Bellaire</td>
</tr>
<tr>
<td>5</td>
<td>Sugarland</td>
</tr>
<tr>
<td>5</td>
<td>Houston</td>
</tr>
</tbody>
</table>
Figure 3.6
One possible database state for the COMPANY relational database schema.

<table>
<thead>
<tr>
<th>WORKS_ON</th>
<th>PROJECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Essn</td>
<td>Pname</td>
</tr>
<tr>
<td>123456789</td>
<td>ProductX</td>
</tr>
<tr>
<td>123456789</td>
<td>ProductY</td>
</tr>
<tr>
<td>666884444</td>
<td>ProductZ</td>
</tr>
<tr>
<td>453453453</td>
<td>Computerization</td>
</tr>
<tr>
<td>453453453</td>
<td>Reorganization</td>
</tr>
<tr>
<td>333445555</td>
<td>Newbenefits</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DEPENDENT</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Essn</td>
<td>Dependent_name</td>
</tr>
<tr>
<td>333445555</td>
<td>Alice</td>
</tr>
<tr>
<td>333445555</td>
<td>Theodore</td>
</tr>
<tr>
<td>333445555</td>
<td>Joy</td>
</tr>
<tr>
<td>987654321</td>
<td>Abner</td>
</tr>
<tr>
<td>987654321</td>
<td>Michael</td>
</tr>
<tr>
<td>888665555</td>
<td>Alice</td>
</tr>
<tr>
<td>123456789</td>
<td>Elizabeth</td>
</tr>
</tbody>
</table>
Figure 4.3
Results of SQL queries when applied to the COMPANY database state shown in Figure 3.6. (a) Q0. (b) Q1. (c) Q2. (d) Q8. (e) Q9. (f) Q10. (g) Q1C.

(a) | Bdate  | Address          |
---|--------|------------------|
    | 1965-01-09 | 731 Fondren, Houston, TX |

(b) | Fname  | Lname  | Address                        |
---|--------|--------|--------------------------------|
    | John   | Smith  | 731 Fondren, Houston, TX       |
    | Franklin | Wong   | 638 Voss, Houston, TX          |
    | Ramesh | Narayan | 975 Fire Oak, Humble, TX        |
    | Joyce | English | 5631 Rice, Houston, TX          |

Query 0. Retrieve the birth date and address of the employee(s) whose name is ‘John B. Smith’.

Q0:  
SELECT Bdate, Address  
FROM EMPLOYEE  
WHERE Fname='John' AND Minit='B' AND Lname='Smith';

Query 1. Retrieve the name and address of all employees who work for the ‘Research’ department.

Q1:  
SELECT Fname, Lname, Address  
FROM EMPLOYEE, DEPARTMENT  
WHERE Dname='Research' AND Dnumber=Dno;
Figure 4.3
Results of SQL queries when applied to the COMPANY database state shown in Figure 3.6. (a) Q0. (b) Q1. (c) Q2. (d) Q8. (e) Q9. (f) Q10. (g) Q1C.

<table>
<thead>
<tr>
<th>Pnumber</th>
<th>Dnum</th>
<th>Lname</th>
<th>Address</th>
<th>Bdate</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>Wallace</td>
<td>291Berry, Bellaire, TX</td>
<td>1941-06-20</td>
</tr>
<tr>
<td>30</td>
<td>4</td>
<td>Wallace</td>
<td>291Berry, Bellaire, TX</td>
<td>1941-06-20</td>
</tr>
</tbody>
</table>

**Query 2.** For every project located in ‘Stafford’, list the project number, the controlling department number, and the department manager’s last name, address, and birth date.

**Q2:**
```
SELECT Pnumber, Dnum, Lname, Address, Bdate
FROM PROJECT, DEPARTMENT, EMPLOYEE
WHERE Dnum=Dnumber AND Mgr_ssn=Ssn AND Plocation='Stafford';
```
TABLES AS SETS IN SQL

- Duplicate tuples may appear in query results
  - From duplicates in base tables
  - From projecting out distinguishing columns
- Keyword `DISTINCT` in the `SELECT` clause eliminates duplicates

**Query 11.** Retrieve the salary of every employee (Q11) and all distinct salary values (Q11A).

Q11:  
```
SELECT ALL Salary 
FROM EMPLOYEE;
```

Q11A:  
```
SELECT DISTINCT Salary 
FROM EMPLOYEE;
```
SET OPERATIONS

- Result treated as a set (no duplicates)
  - **UNION, EXCEPT** (difference), **INTERSECT**

- Corresponding multiset (bag) operations:
  - **UNION ALL, EXCEPT ALL, INTERSECT ALL**

- Arguments must be *union-compatible*
  - Same number of columns
  - Corresponding columns of same type

---

**Query 4.** Make a list of all project numbers for projects that involve an employee whose last name is ‘Smith’, either as a worker or as a manager of the department that controls the project.

Q4A: 

```
(SELECT DISTINCT Pnumber
  FROM PROJECT, DEPARTMENT, EMPLOYEE
  WHERE Dnum=Dnumber AND Mgr_ssn=Ssn
    AND Lname='Smith')
UNION
(SELECT DISTINCT Pnumber
  FROM PROJECT, WORKS_ON, EMPLOYEE
  WHERE Pnumber=Pno AND Essn=Ssn
    AND Lname='Smith');
```
OTHER OPERATORS

- Standard arithmetic operators:
  - Addition (+), subtraction (−), multiplication (⋅), and division (⁄)
- [NOT] LIKE comparison operator
  - Used for string pattern matching
  - Percent sign (%) matches zero or more characters
  - Underscore (_) matches a single character
  e.g., to also match Tommy Lee Jones as supporting actor:

```
SELECT award, actor, persona, Role.movie
FROM Honours, Role
WHERE category LIKE '%%actor' AND winner = actor
  AND Honours.movie = Role.movie;
```

- [NOT] BETWEEN comparison operator
  - WHERE year BETWEEN 1990 AND 2010
  equivalent to WHERE year >= 1990 AND YEAR <= 2010
LECTURE SUMMARY

- Introduction to SQL
  - Comprehensive language
  - Data definition including constraint specification
  - Basic SELECT-FROM-WHERE
  - Set operators