# Database Programming

## SQL

## (Structured Query Language)

Edison Nkurunungi
+256 772 955 373/ 702 955 372
enkurunungi@uict.ac.ug

# Chapter 3 - Objectives

- Purpose and importance of SQL.
- How to retrieve data using SELECT.
- How to insert data using INSERT.
- How to update data using UPDATE.
- How to delete data using DELETE.
- How to create new tables using CREATE TABLE.
- About an alternative language, QBE.

# SQL

- Main language for relational DBMSs.
- Main characteristics:
  - relatively easy to learn;
  - non-procedural - you specify *what* information you require, rather than *how* to get it;
  - essentially free-format;
  - consists of standard English words like SELECT, INSERT, and UPDATE;
  - can be used by range of users.

# Importance of SQL

- First and, so far, only standard database language to gain widespread acceptance.
- Huge investment from both vendors and users.
  - It is used in many different types of DBMS such as MySQL, Oracle, DB2, Microsoft SQL Server, and Microsoft Access.
  - In this course, students will learn how to use SQL to retrieve, store or update data in MySQL.

# Objectives of SQL

- Ideally database language should let user:
  - create database and table structures.
  - perform basic tasks like insert, update, delete;
  - perform both simple and complex queries.
- Must perform these tasks with minimal user effort.
- Must be easy to learn.

# Objectives of SQL

- SQL is a database language with 2 major components:
  - a DDL for defining database structure;
  - a DML for retrieving and updating data.
- SQL can be used interactively or embedded in a high-level language (eg. C, C++).

# Writing SQL Commands

- SQL statement consists of *reserved words* and *user-defined words*.
  - Reserved words: fixed part of SQL and must be spelt exactly as required and cannot be split across lines.
  - User-defined words: made up by user and represent names of various database objects such as tables, columns, views.

# Writing SQL Commands

- Most components of an SQL statement are *case insensitive*, except for literal character data.
- More readable with indentation and lineation:
- Each clause should begin on a new line.
- Start of a clause should line up with start of other clauses

# Literals

- Literals are constants used in SQL statements.
- All non-numeric literals must be enclosed in single quotes (eg. ‘London’).
- All numeric literals must not be enclosed in quotes (eg. 650.00).

# SQL Identifiers

SQL identifiers are used to identify objects in the database such as table names, view names and columns. The following restrictions (syntax rules) are imposed on an identifier

# SQL Syntax Rules

The most basic rules of SQL are:

- Identifiers (names of tables, columns, and other objects) should contain between 1 and 30 characters.  The identifiers can be upper or lower case, but no embedded spaces are allowed.
- For example, WORK PHONE would have to be written as WORKPHONE or WORK_PHONE.

# SQL Syntax Rules

- SQL is not case sensitive, although SQL keywords such as SELECT or FROM are usually capitalized.
- Keywords have predefined meanings and cannot be used as identifiers.

# SQL Syntax Rules

- SQL statements can take up more than one line (and there are no restrictions on the number of words per line or where to break a line). However, a new line is often started when a new clause in an SQL statement begins.
- Commands begin with the SQL element (e.g. CREATE or SELECT).

# Semicolon after SQL Statements?

- Some database systems require a semicolon at the end of each SQL statement.
- Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.
- When we are using for example MS Access and SQL Server 2000 we do not have to put a semicolon after each SQL statement, but some database programs force you to use it.

# SQL Syntax Elements

**DDL STATEMENTS**

The following described keywords identify commonly used SQL DDL verbs:

- **CREATE** Creates a database, table, view, etc.
- **DROP** Removes a database, table, view, etc.
- **ALTER** Alters a database, table, etc.

# SQL Syntax Elements

**DML Statements**

The following described keywords identify commonly used SQL DML verbs:

- **SELECT** Retrieves the specified records.
- **UPDATE** Changes values in the specified rows.
- **INSERT** Adds a new row.
- **DELETE** Removes the specified rows.

# SQL Syntax Elements

**DCL Statements**

The following described keywords identify commonly used SQL DCL verbs:

- **GRANT** Grants or gives privileges to users.
- **REVOKE** Revokes or takes away privileges that were granted to users with the GRANT statement.

# Privileges

**Privileges** are the actions that a user is permitted to carry out on a given base table or view. The privileges defined by the ISO standards are:

- **SELECT** — Permits the user or object to SELECT data from the table or view.
- **INSERT** — Permits the user or object to INSERT rows into the table or view.
- **UPDATE** — Permits the user or object to UPDATE rows in the table or view, optionally restricted to specific columns.

# Privileges

- **DELETE** — Permits the user or object to DELETE rows from the table or view.
- **REFERENCES** — Permits the user or object to reference the specified column[s] of the table via a foreign key. If the primary or unique key referenced by the foreign key of the other table is composite then all columns of the key must be specified.

# SELECT Statement

```
SELECT [DISTINCT | ALL]
{* | [columnExprn [AS newName]] [,...] }
FROM TableName [alias] [, ...]
[WHERE condition]
[GROUP BY columnList] [HAVING condition]
[ORDER BYcolumnList]
```

# SELECT Statement

| | |
|---|---|
| SELECT | Specifies which columns are to appear in output. |
| FROM | Specifies table(s) to be used. |
| WHERE | Filters rows. |
| GROUP BY | Forms groups of rows with same column value. |
| HAVING | Filters groups subject to some condition. |
| ORDER BY | Specifies the order of the output. |

# SELECT Statement

- Order of the clauses cannot be changed.
- Only SELECT and FROM are mandatory.

# SQL Operators

The following described basic operators specify conditions and perform logical and numeric functions:

| | |
|---|---|
| AND | Both conditions must be met |
| OR | At least one condition must be met |
| NOT | Exclude the condition following |
| LIKE | Matches with a pattern |
| IN | Matches with a list of values |
| BETWEEN | Matches with a range of values |

# OPERATORS

| | |
|---|---|
| = | Equal to |
| <> | Not equal to |
| < | Less than |
| > | Greater than |
| <= | Less than or equal to |
| >= | Greater than or equal to |

# OPERATORS

| | |
|---|---|
| + | Addition |
| − | Subtraction |
| / | Division |
| * | Multiplication |

# Quotes Around Text Fields

- SQL uses single quotes around text values (most database systems will also accept double quotes).
- However, numeric values should not be enclosed in quotes.

For text values:

- This is correct:
  SELECT * FROM Persons WHERE FirstName='Tove‘;

# Quotes Around Text Fields

- This is wrong:
  SELECT * FROM Persons WHERE FirstName=Tove;

For numeric values:

- This is correct:
  SELECT * FROM Persons WHERE Year=1965;

  This is wrong:
- SELECT * FROM Persons WHERE Year='1965‘;

# EXAMPLES: IMPLEMENTATION OF SQL

**Creating A Database**:

**SYNTAX:** CREATE DATABASE database_name

e.g: **CREATE DATABASE Records;**

- **Note: Read about SQL data types**

# EXAMPLES: IMPLEMENTATION OF SQL

**CREATE TABLE**

**SYTAX:**

```
CREATE TABLE table_name
(     field1 datatype [ NOT NULL ],
      field2 datatype [ NOT NULL ],
      field3 datatype [ NOT NULL ]...)
```

e.g

```
CREATE TABLE BILLS(
     NAME CHAR(30) NOT NULL,
     AMOUNT NUMBER,
     ACCOUNT_ID  char(20) NOT NULL);
```

# PRACTICE EXERCISES
## Basic Table Creation

```
CREATE TABLE Products
  (
    prod_id      CHAR(10)          NOT NULL,
    vend_id      CHAR(10)          NOT NULL,
    prod_name    CHAR(254)         NOT NULL,
    prod_price   DECIMAL(8,2)      NOT NULL,
    prod_desc    VARCHAR(1000)
  );
```

# PRACTICE EXERCISES
# Basic Table Creation

```
CREATE TABLE Vendors
(
    vend_id          CHAR(10)    NOT NULL,
    vend_name        CHAR(50)    NOT NULL,
    vend_address     CHAR(50) ,
    vend_city        CHAR(50) ,
    vend_state       CHAR(5)    ,
    vend_zip         CHAR(10)   ,
    vend_country     CHAR(50)
);
```

# INSERT values into a table

```
INSERT INTO TableName [ (columnList) ]
VALUES (dataValueList)
```

- *columnList* is optional; if omitted, SQL assumes a list of all columns in their original CREATE TABLE order.
- Any columns omitted must have been declared as NULL or a DEFAULT was specified when table was created.

# INSERT values into a table

- *dataValueList* must match *columnList* as follows:
  - number of items in each list must be same;
  - must be direct correspondence in position of items in two lists;
  - data type of each item in *dataValueList* must be compatible with data type of corresponding column.

# 3.19 INSERT values into a table

Insert a row into the Video table.

```
INSERT INTO Video
VALUES (‘207132’, ‘Die Another Day’,
   ‘Action’ 5.00, 21.99, ‘D1001’ );
```

Table name

Attribute names

# Tables in SQL

Product

| PName | Price | Category | Manufacturer |
|---|---|---|---|
| Gizmo | 19.99 | Gadgets | GizmoWorks |
| Powergizmo | 29.99 | Gadgets | GizmoWorks |
| SingleTouch | 149.99 | Photography | Canon |
| MultiTouch | 203.99 | Household | Hitachi |

Tuples or rows

# Table Explained

- The *schema* of a table is the table name and its attributes:

Product(PName, Price, Category, Manfacturer)

- A *key* is an attribute whose values are unique;
  we underline a key

Product(<u>PName</u>, Price, Category, Manfacturer)

# SQL Query

Basic form: (plus many many more bells and whistles)

```
SELECT  <attributes>
FROM    <one or more relations>
WHERE   <conditions>
```

# Simple SQL Query

Product

| PName | Price | Category | Manufacturer |
|---|---|---|---|
| Gizmo | $19.99 | Gadgets | GizmoWorks |
| Powergizmo | $29.99 | Gadgets | GizmoWorks |
| SingleTouch | $149.99 | Photography | Canon |
| MultiTouch | $203.99 | Household | Hitachi |

```
SELECT   *
FROM     Product
WHERE    category='Gadgets'
```

| PName | Price | Category | Manufacturer |
|---|---|---|---|
| Gizmo | $19.99 | Gadgets | GizmoWorks |
| Powergizmo | $29.99 | Gadgets | GizmoWorks |

"selection"

# Simple SQL Query

Product

| PName | Price | Category | Manufacturer |
|---|---|---|---|
| Gizmo | $19.99 | Gadgets | GizmoWorks |
| Powergizmo | $29.99 | Gadgets | GizmoWorks |
| SingleTouch | $149.99 | Photography | Canon |
| MultiTouch | $203.99 | Household | Hitachi |

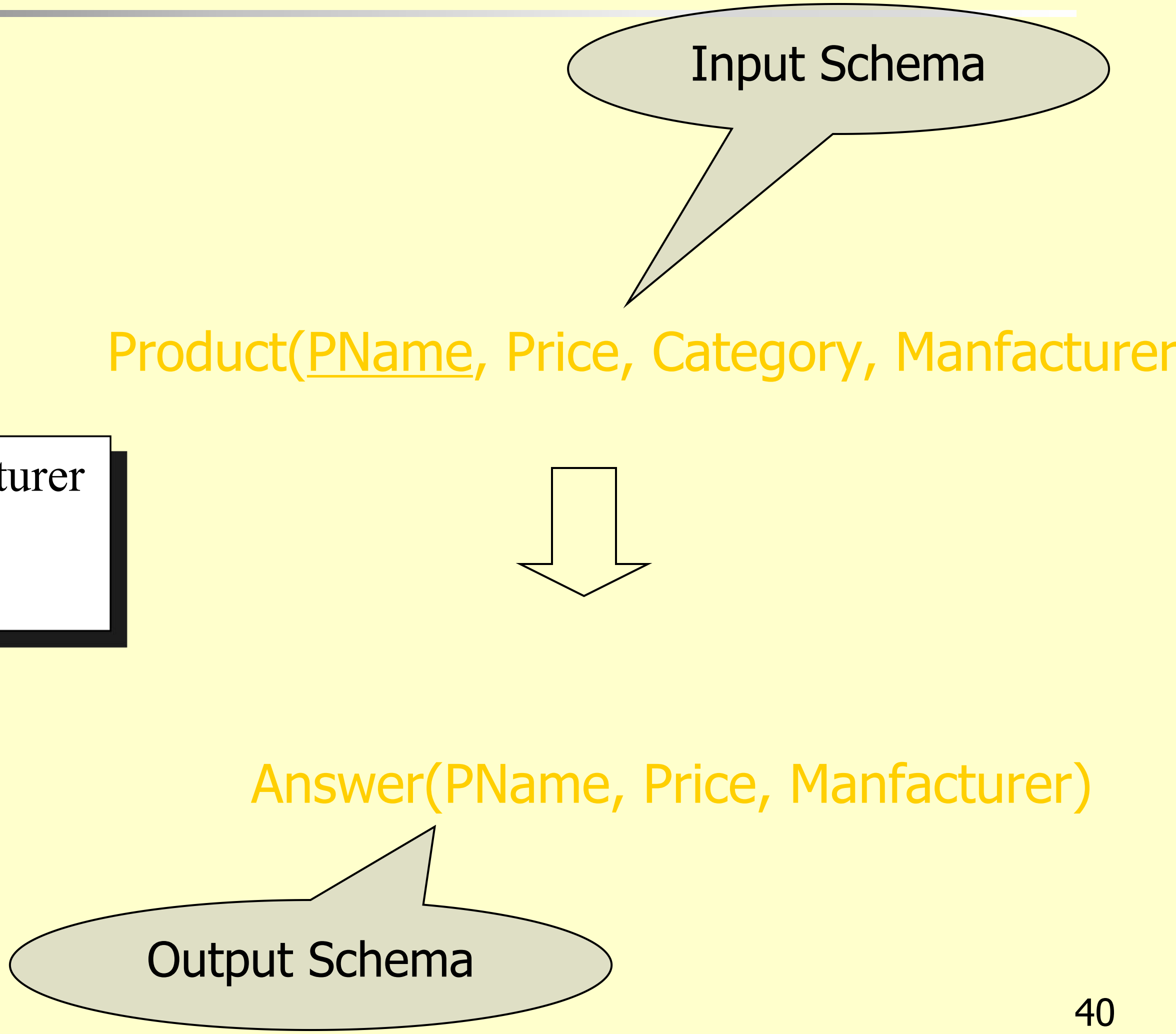
```
SELECT   PName, Price, Manufacturer
FROM     Product
WHERE    Price > 100
```

| PName | Price | Manufacturer |
|---|---|---|
| SingleTouch | $149.99 | Canon |
| MultiTouch | $203.99 | Hitachi |

"selection" and "projection"

# Notation

```
SELECT   PName, Price, Manufacturer
FROM     Product
WHERE    Price > 100;
```



Input Schema

Product(<u>PName</u>, Price, Category, Manfacturer

Answer(PName, Price, Manfacturer)

Output Schema

# 3.1 All Columns, All Rows

List full details of all videos.

```
SELECT catalogNo, title, category,
           dailyRental, price, directorNo
FROM Video;
```

- Can use * as an abbreviation for ‘all columns’:

```
SELECT *
FROM Video;
```

# 3.1 All Columns, All Rows

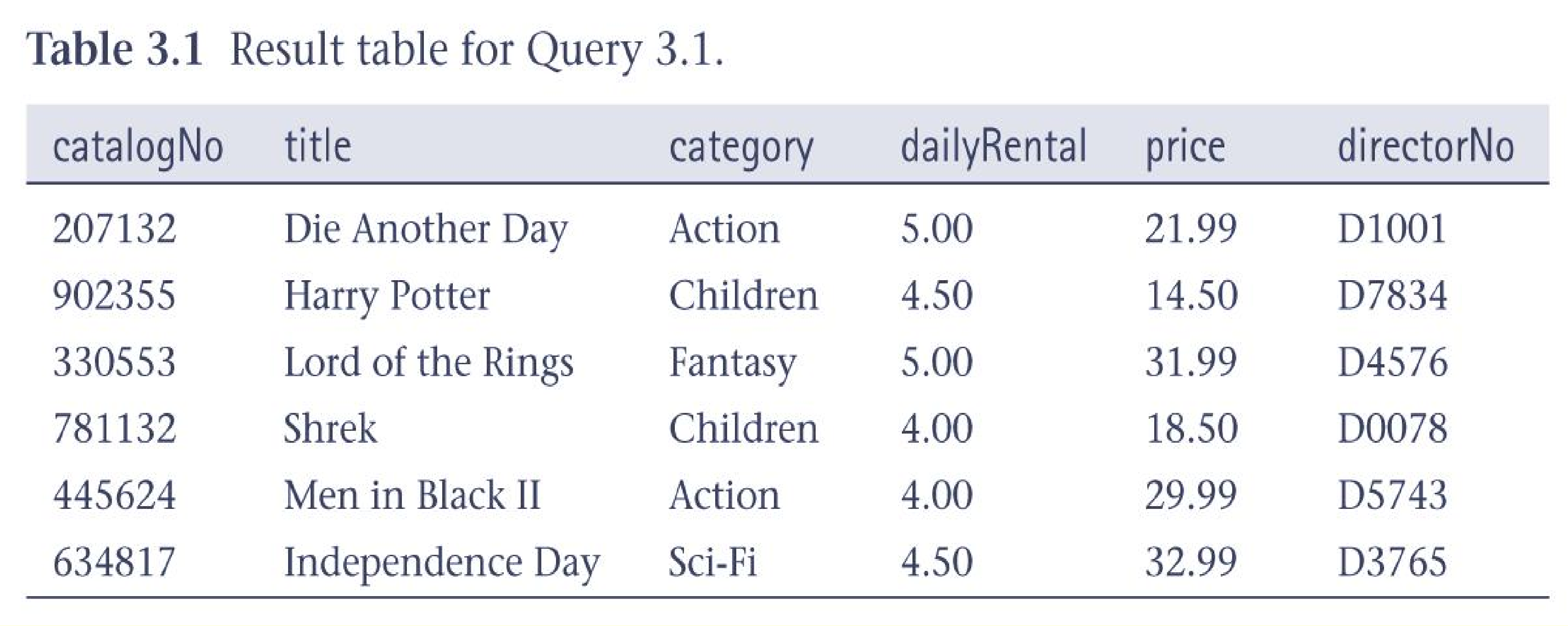


Table 3.1 Result table for Query 3.1.

| catalogNo | title | category | dailyRental | price | directorNo |
|---|---|---|---|---|---|
| 207132 | Die Another Day | Action | 5.00 | 21.99 | D1001 |
| 902355 | Harry Potter | Children | 4.50 | 14.50 | D7834 |
| 330553 | Lord of the Rings | Fantasy | 5.00 | 31.99 | D4576 |
| 781132 | Shrek | Children | 4.00 | 18.50 | D0078 |
| 445624 | Men in Black II | Action | 4.00 | 29.99 | D5743 |
| 634817 | Independence Day | Sci-Fi | 4.50 | 32.99 | D3765 |

# 3.2 Specific Columns, All Rows

List the catalog number, title and daily rental rate of all videos.

```
SELECT catalogNo, title, dailyRental
FROM Video;
```

# 3.2 Specific Columns, All Rows

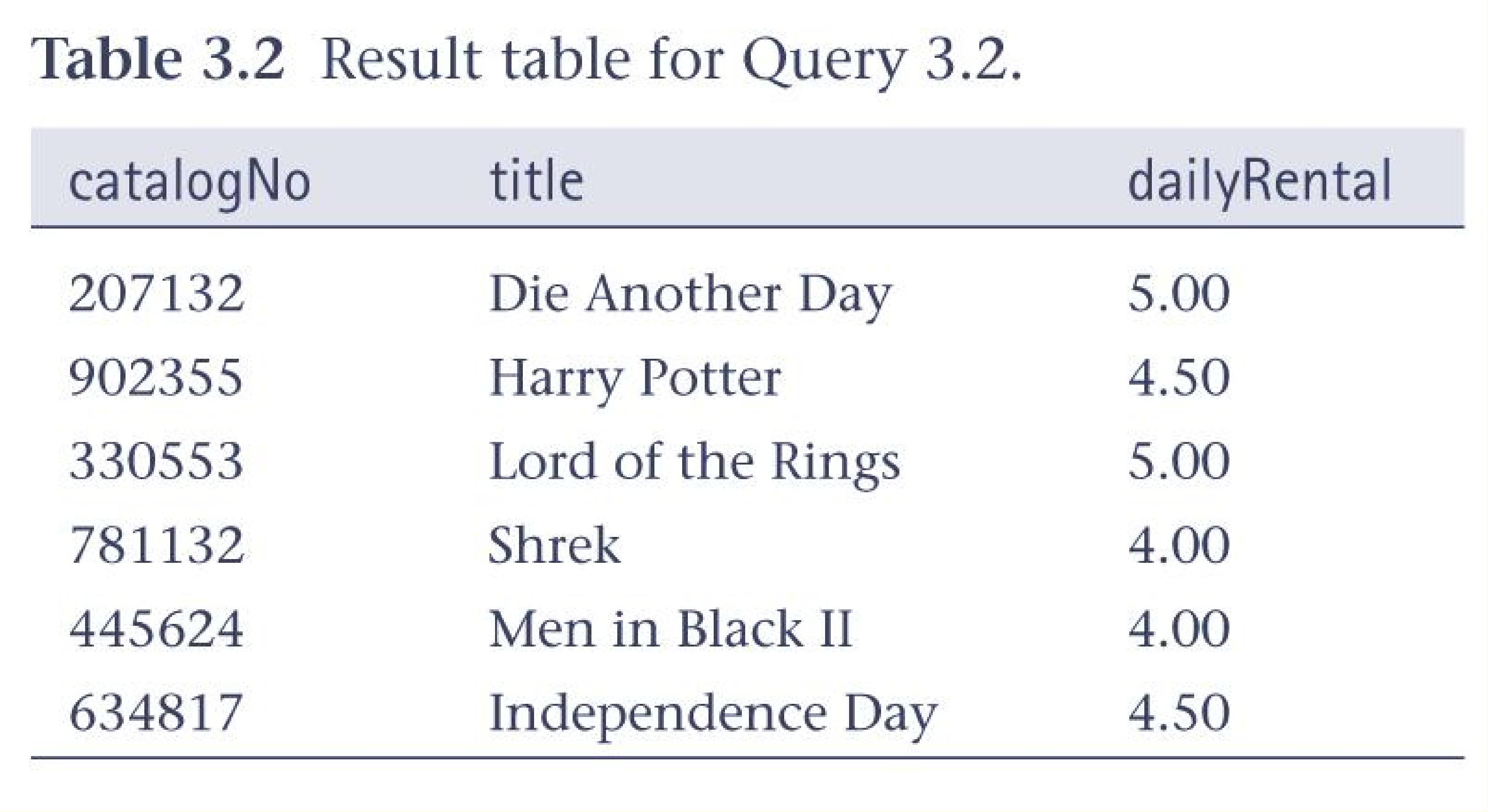


Table 3.2 Result table for Query 3.2.

| catalogNo | title | dailyRental |
|---|---|---|
| 207132 | Die Another Day | 5.00 |
| 902355 | Harry Potter | 4.50 |
| 330553 | Lord of the Rings | 5.00 |
| 781132 | Shrek | 4.00 |
| 445624 | Men in Black II | 4.00 |
| 634817 | Independence Day | 4.50 |

# 3.3 Use of DISTINCT

**List all video categories.**

```
SELECT category
FROM Video;
```

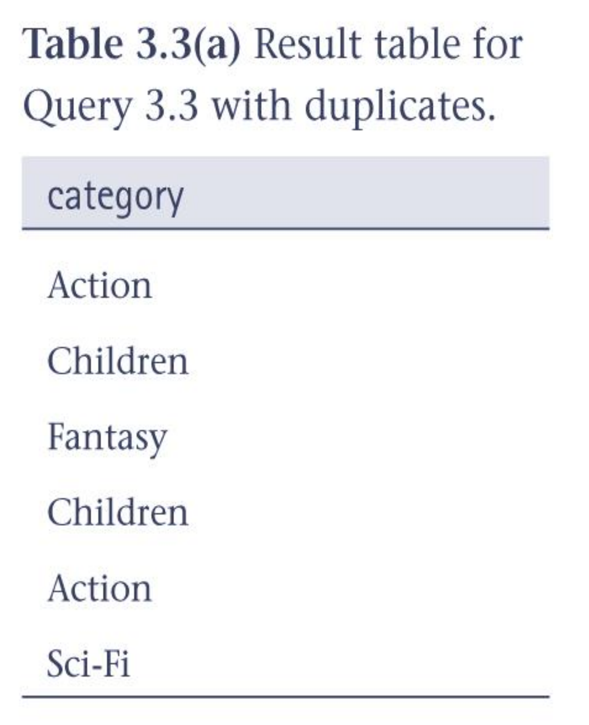


**Table 3.3(a)** Result table for Query 3.3 with duplicates.

| category |
|---|
| Action |
| Children |
| Fantasy |
| Children |
| Action |
| Sci-Fi |

# 3.3 Use of DISTINCT

- **Use DISTINCT to eliminate duplicates:**

```
SELECT DISTINCT category
FROM Video;
```

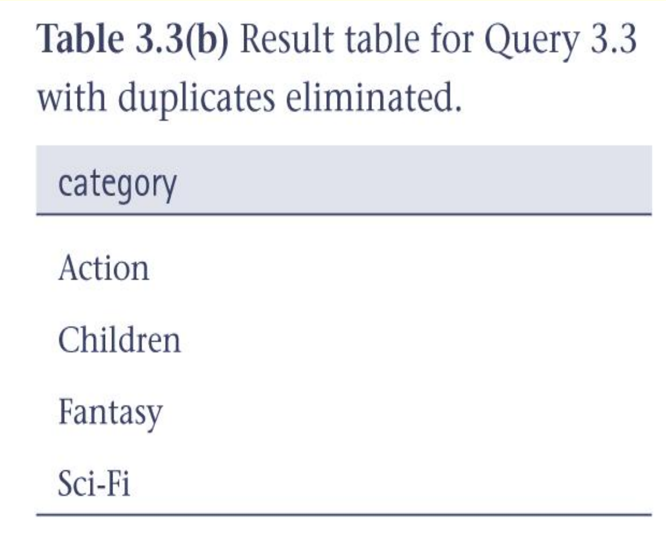


**Table 3.3(b)** Result table for Query 3.3 with duplicates eliminated.

| category |
|---|
| Action |
| Children |
| Fantasy |
| Sci-Fi |

# 3.4 Calculated Fields

List rate for renting videos for 3 days.

```
SELECT catalogNo, title, dailyRental*3
FROM Video;
```

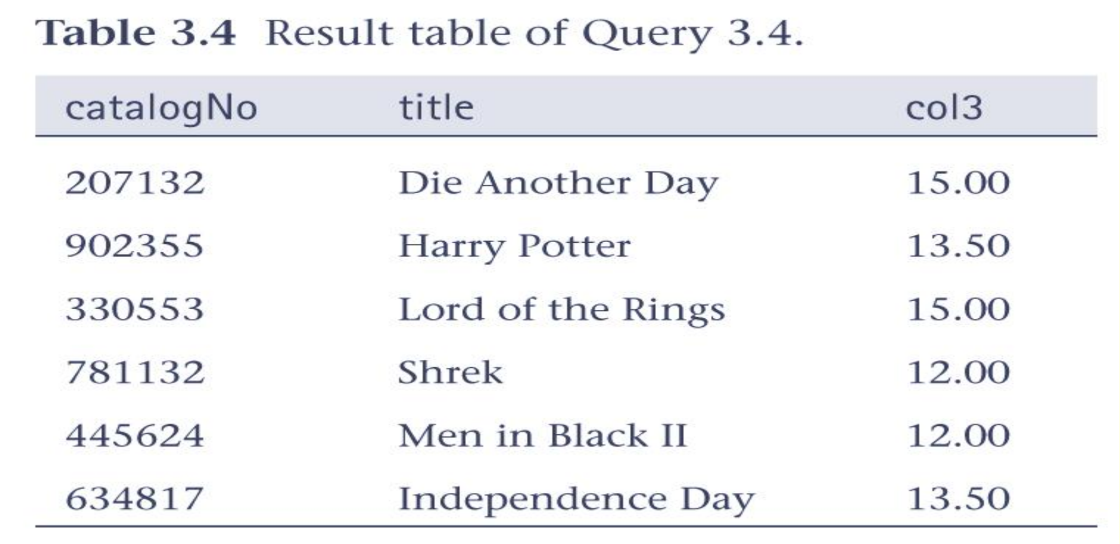


**Table 3.4** Result table of Query 3.4.

| catalogNo | title | col3 |
|---|---|---|
| 207132 | Die Another Day | 15.00 |
| 902355 | Harry Potter | 13.50 |
| 330553 | Lord of the Rings | 15.00 |
| 781132 | Shrek | 12.00 |
| 445624 | Men in Black II | 12.00 |
| 634817 | Independence Day | 13.50 |

# 3.4 Calculated Fields

- To name column, use AS clause:

```
SELECT catalogNo, title,
            dailyRental*3 AS threeDayRate
FROM Video;
```

# 3.5 Comparison Search Condition

**List all staff with a salary greater than $10,000.**

```
SELECT staffNo, name, position, salary
FROM Staff
WHERE salary > 10000;
```

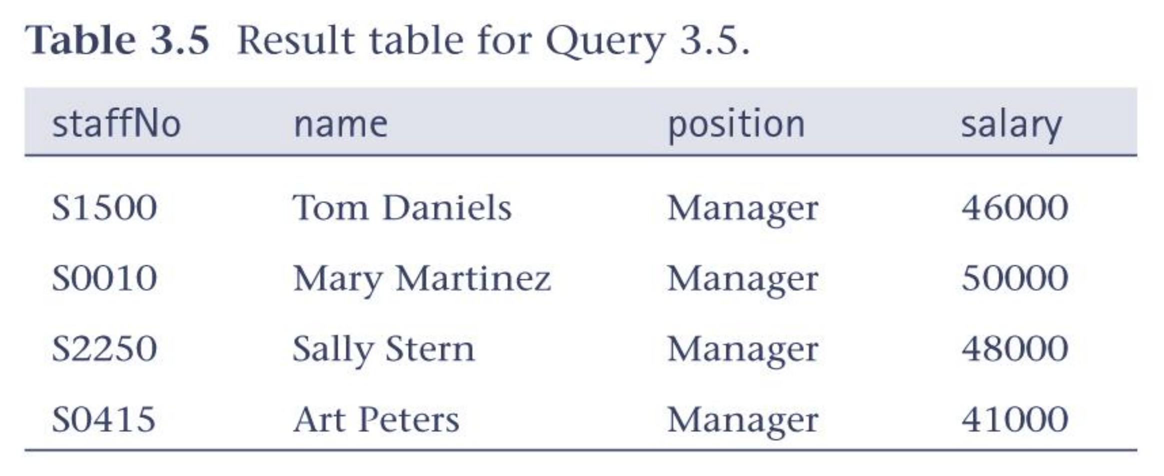


**Table 3.5** Result table for Query 3.5.

| staffNo | name | position | salary |
|---|---|---|---|
| S1500 | Tom Daniels | Manager | 46000 |
| S0010 | Mary Martinez | Manager | 50000 |
| S2250 | Sally Stern | Manager | 48000 |
| S0415 | Art Peters | Manager | 41000 |

# 3.6 Range Search Condition

List all staff with a salary between $45,000 and $50,000.

```
SELECT staffNo, name, position, salary
FROM Staff
WHERE salary BETWEEN 45000 AND 50000;
```

- BETWEEN test includes the endpoints of range.

# 3.6 Range Search Condition

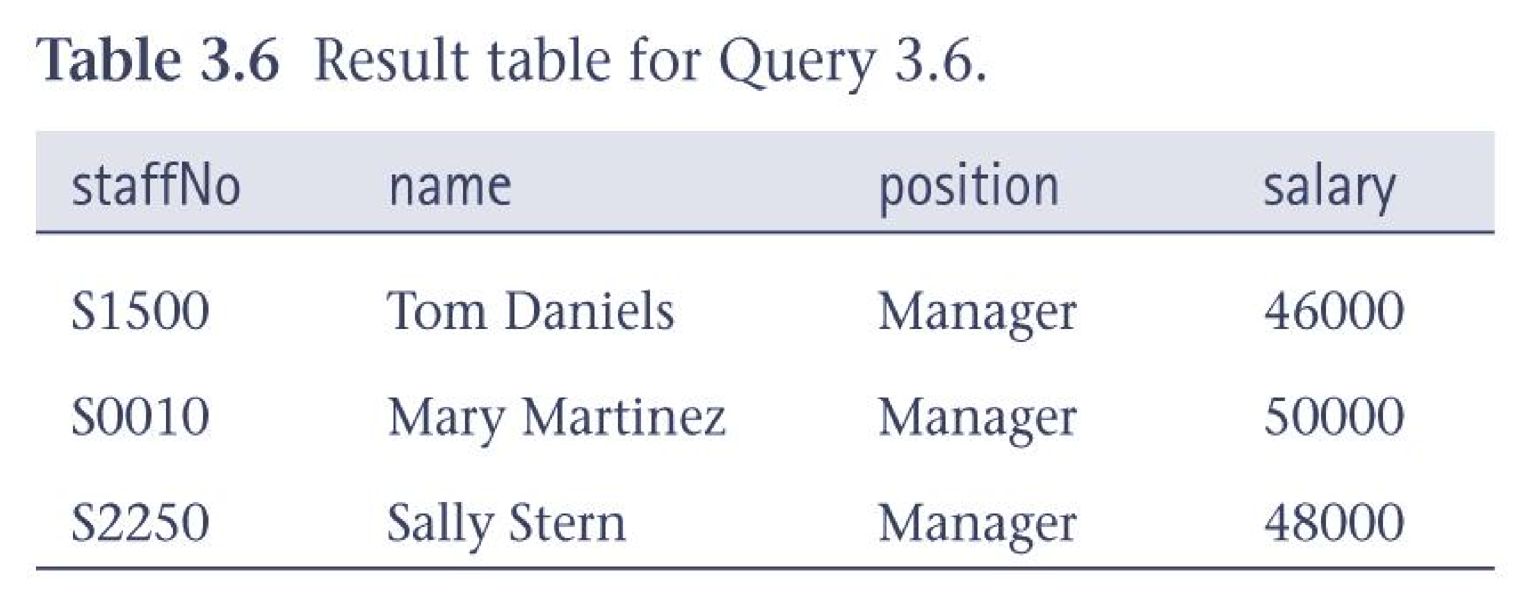


**Table 3.6** Result table for Query 3.6.

| staffNo | name | position | salary |
|---|---|---|---|
| S1500 | Tom Daniels | Manager | 46000 |
| S0010 | Mary Martinez | Manager | 50000 |
| S2250 | Sally Stern | Manager | 48000 |

# 3.6 Range Search Condition

- Also a negated version NOT BETWEEN.
- BETWEEN does not add much to SQL's expressive power. Could also write:

```
SELECT staffNo, name, position, salary
FROM Staff
WHERE salary >= 45000 AND salary <= 50000;
```

- Useful, though, for a range of values.

# 3.7 Set Membership

List all videos in the Action and Children categories.

```
SELECT catalogNo, title, category
FROM Video
WHERE category IN (‘Action’, ‘Children’);
```

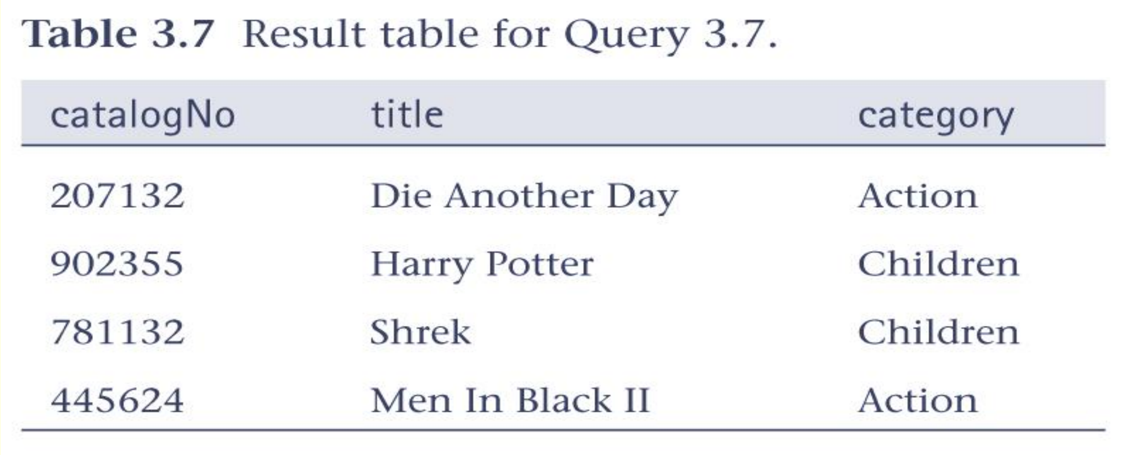


**Table 3.7** Result table for Query 3.7.

| catalogNo | title | category |
|---|---|---|
| 207132 | Die Another Day | Action |
| 902355 | Harry Potter | Children |
| 781132 | Shrek | Children |
| 445624 | Men In Black II | Action |

# 3.7 Set Membership

- There is a negated version (NOT IN).
- IN does not add much to SQL's expressive power. Could have expressed this as:

```
SELECT catalogNo, title, category
FROM Video
WHERE category =‘Action’ OR category =‘Children’
```

- IN is more efficient when set contains many values.

# 3.8 Pattern Matching

List all staff whose first name is Sally.

```
SELECT staffNo, name, position, salary
FROM Staff
WHERE name LIKE 'Sally%';
```

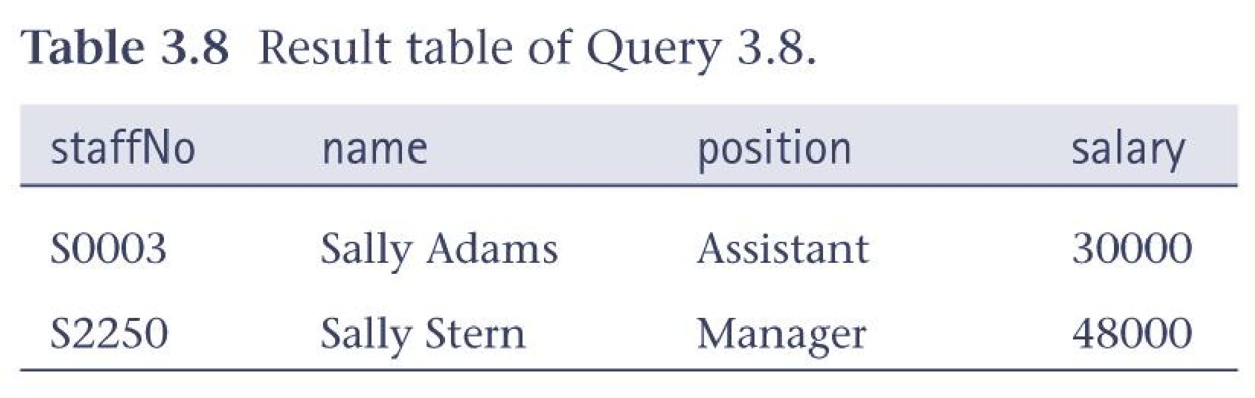


Table 3.8 Result table of Query 3.8.

| staffNo | name | position | salary |
|---|---|---|---|
| S0003 | Sally Adams | Assistant | 30000 |
| S2250 | Sally Stern | Manager | 48000 |

# 3.8 Pattern Matching

- SQL has two special pattern matching symbols:
  - %: sequence of zero or more characters;
  - _ (underscore): any single character.
- LIKE ‘Sally%’ means the first 5 characters must be Sally followed by anything.

# 3.9 NULL Search Condition

List the video rentals that have not yet been returned.

- Have to test for null explicitly using special keyword IS NULL:

```
SELECT dateOut, memberNo, videoNo
FROM RentalAgreement
WHERE dateReturn IS NULL;
```

# 3.9 NULL Search Condition

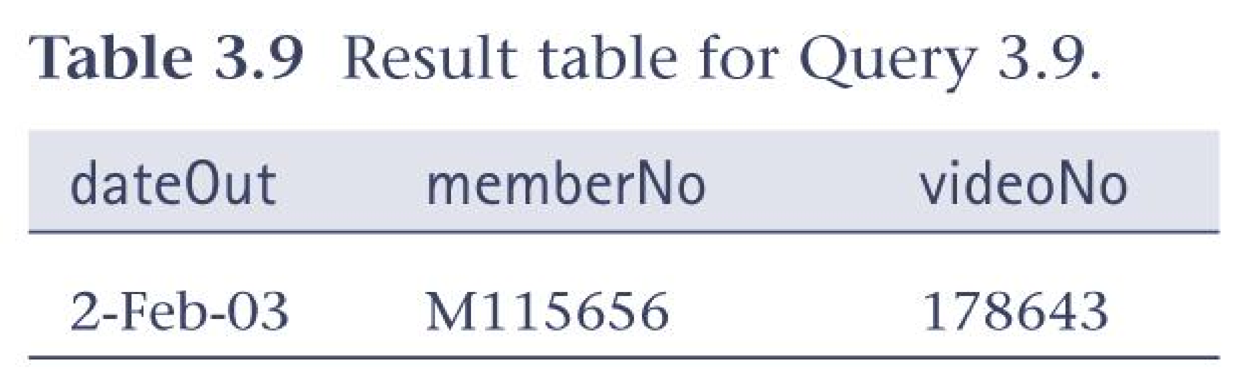


Table 3.9 Result table for Query 3.9.

| dateOut | memberNo | videoNo |
|---|---|---|
| 2-Feb-03 | M115656 | 178643 |

- **Negated version (IS NOT NULL) can test for non-null values.**

# 3.10 Single Column Ordering

List all videos in descending order of price.

```
SELECT *
FROM Video
ORDER BY price DESC;
```

# 3.10 Single Column Ordering

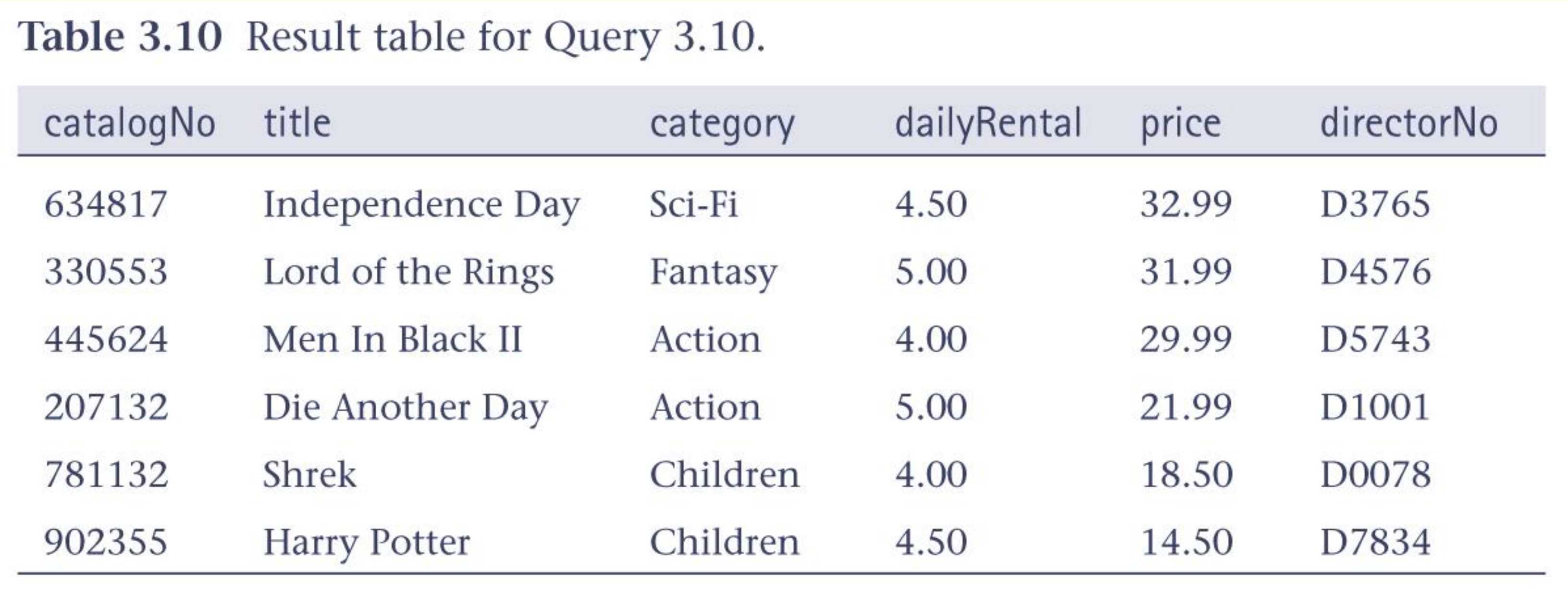


**Table 3.10** Result table for Query 3.10.

| catalogNo | title | category | dailyRental | price | directorNo |
|---|---|---|---|---|---|
| 634817 | Independence Day | Sci-Fi | 4.50 | 32.99 | D3765 |
| 330553 | Lord of the Rings | Fantasy | 5.00 | 31.99 | D4576 |
| 445624 | Men In Black II | Action | 4.00 | 29.99 | D5743 |
| 207132 | Die Another Day | Action | 5.00 | 21.99 | D1001 |
| 781132 | Shrek | Children | 4.00 | 18.50 | D0078 |
| 902355 | Harry Potter | Children | 4.50 | 14.50 | D7834 |

# SELECT Statement - Aggregates

- ISO SQL defines five aggregate functions:

COUNT returns number of values in specified column.

SUM returns sum of values in specified column.

AVG returns average of values in specified column.

MIN returns smallest value in specified column.

MAX returns largest value in specified column.

# SELECT Statement - Aggregates

- Each operates on a single column of a table and returns a single value.
- COUNT, MIN, and MAX apply to numeric and non-numeric fields, but SUM and AVG only for numeric fields.
- Apart from COUNT(*), each function eliminates nulls first and operates only on remaining non-null values.

# SELECT Statement - Aggregates

- COUNT(*) counts all rows of a table, regardless of whether nulls or duplicate values occur.
- Can use DISTINCT before column name to eliminate duplicates.
- DISTINCT has no effect with MIN/MAX, but may have with SUM/AVG.

# SELECT Statement - Aggregates

- Aggregate functions can be used only in SELECT list and in HAVING clause.
- If SELECT list includes an aggregate function and there is no GROUP BY clause, SELECT list cannot reference a column out with an aggregate function.
- For example, following is illegal:

```
SELECT staffNo, COUNT(salary)
FROM Staff;
```

# 3.11 Use of COUNT and SUM

List total number of staff with salary greater than $40,000 and the sum of their salaries.

```
SELECT COUNT(staffNo) AS totalStaff,
            SUM(salary) as totalSalary
FROM Staff
WHERE salary > 40000;
```

# 3.11 Use of COUNT and SUM

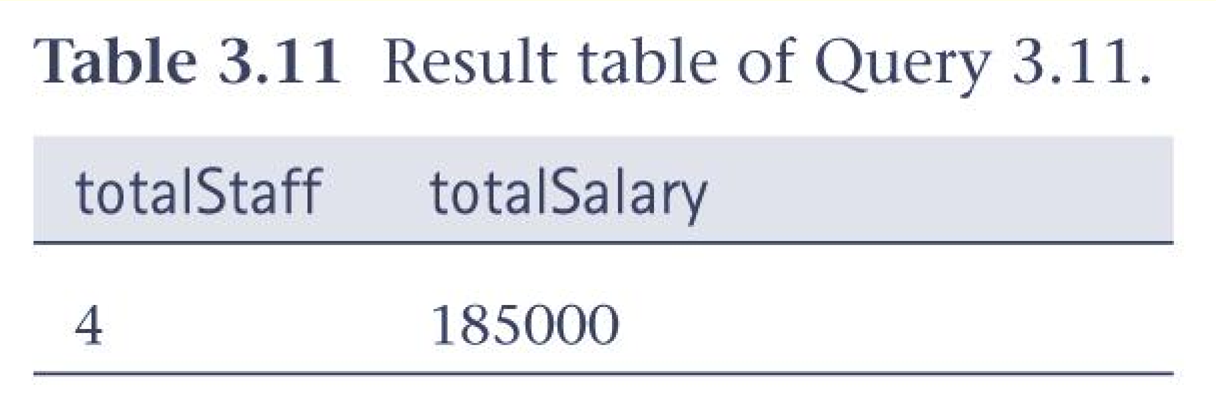


Table 3.11 Result table of Query 3.11.

| totalStaff | totalSalary |
|---|---|
| 4 | 185000 |

# 3.12 Use of MIN, MAX and AVG

**List the minimum, maximum, and average staff salary.**

```
SELECT MIN(salary) AS minSalary,
          MAX(salary) AS maxSalary,
          AVG(salary) AS avgSalary
FROM Staff;
```

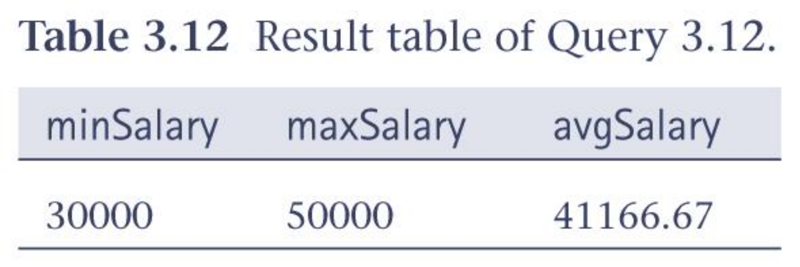


**Table 3.12** Result table of Query 3.12.

| minSalary | maxSalary | avgSalary |
|---|---|---|
| 30000 | 50000 | 41166.67 |

# SELECT Statement - Grouping

- Use GROUP BY clause to get sub-totals.
- SELECT and GROUP BY closely integrated: each item in SELECT list must be *single-valued per group*, and SELECT clause may only contain:
  - column names
  - aggregate functions
  - constants
  - expression with combination of above.

# SELECT Statement - Grouping

- All column names in SELECT list must appear in GROUP BY clause unless used only in an aggregate function.
- If used, WHERE is applied first, then groups are formed from remaining rows satisfying predicate.
- ISO considers two nulls to be equal for purposes of GROUP BY.

# 3.13 Use of GROUP BY

Find number of staff in each branch and sum of their salaries.

```
SELECT branchNo,
            COUNT(staffNo) AS totalStaff,
            SUM(salary) AS totalSalary
FROM Staff
GROUP BY branchNo
ORDER BY branchNo;
```

# 3.13 Use of GROUP BY

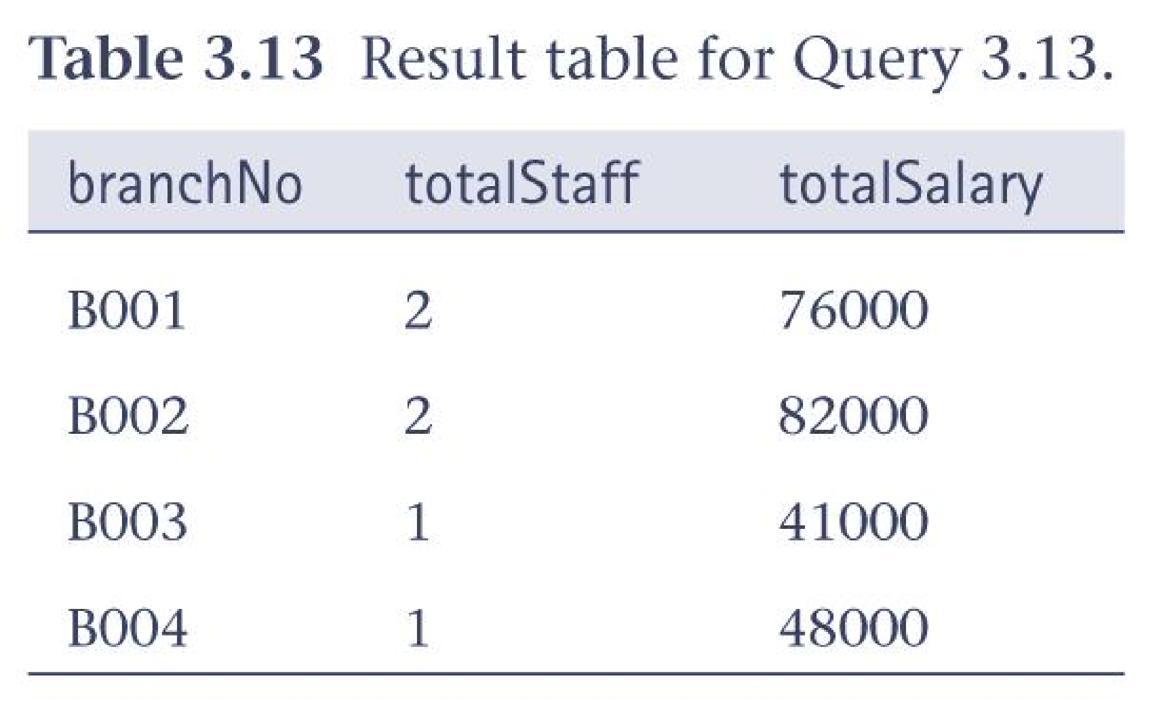


Table 3.13 Result table for Query 3.13.

| branchNo | totalStaff | totalSalary |
|---|---|---|
| B001 | 2 | 76000 |
| B002 | 2 | 82000 |
| B003 | 1 | 41000 |
| B004 | 1 | 48000 |

# Restricted Groupings - HAVING clause

- HAVING clause designed for use with GROUP BY to restrict groups that appear in final result table.
- Similar to WHERE, but WHERE filters individual rows whereas HAVING filters groups.
- Column names in HAVING clause must also appear in the GROUP BY list or be contained within an aggregate function.

# 3.14 Use of HAVING

For each branch with more than 1 member of staff, find number of staff in each branch and sum of their salaries.

```
SELECT branchNo,
        COUNT(staffNo) AS totalStaff,
         SUM(salary) AS totalSalary
FROM Staff
GROUP BY branchNo
HAVING COUNT(staffNo) > 1
ORDER BY branchNo;
```

# 3.14 Use of HAVING

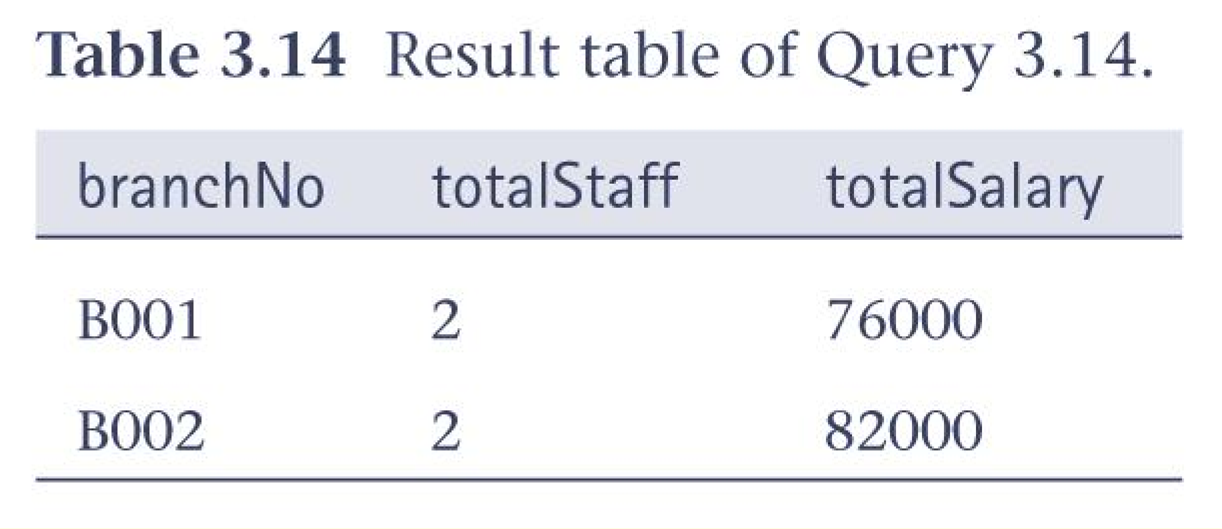


Table 3.14 Result table of Query 3.14.

| branchNo | totalStaff | totalSalary |
|---|---|---|
| B001 | 2 | 76000 |
| B002 | 2 | 82000 |

# Subqueries

- Some SQL statements can have a SELECT embedded within them.
- A subselect can be used in WHERE and HAVING clauses of an outer SELECT, where it is called a *subquery* or *nested query*.
- Subselects may also appear in INSERT, UPDATE, and DELETE statements.

# 3.15 Subquery with Equality

Find staff who work in branch at ‘8 Jefferson Way’.

```
SELECT staffNo, name, position
FROM Staff
WHERE branchNo =
        (SELECT branchNo
         FROM Branch
         WHERE street=‘8 Jefferson Way’);
```

# 3.15 Subquery with Equality

- Inner SELECT finds branch number for branch at ‘8 Jefferson Way’ (‘B001’).
- Outer SELECT then retrieves details of all staff who work at this branch.
- Outer SELECT then becomes:

```
SELECT staffNo, name, position
FROM Staff
WHERE branchNo = ‘B001’;
```

# 3.15 Subquery with Equality

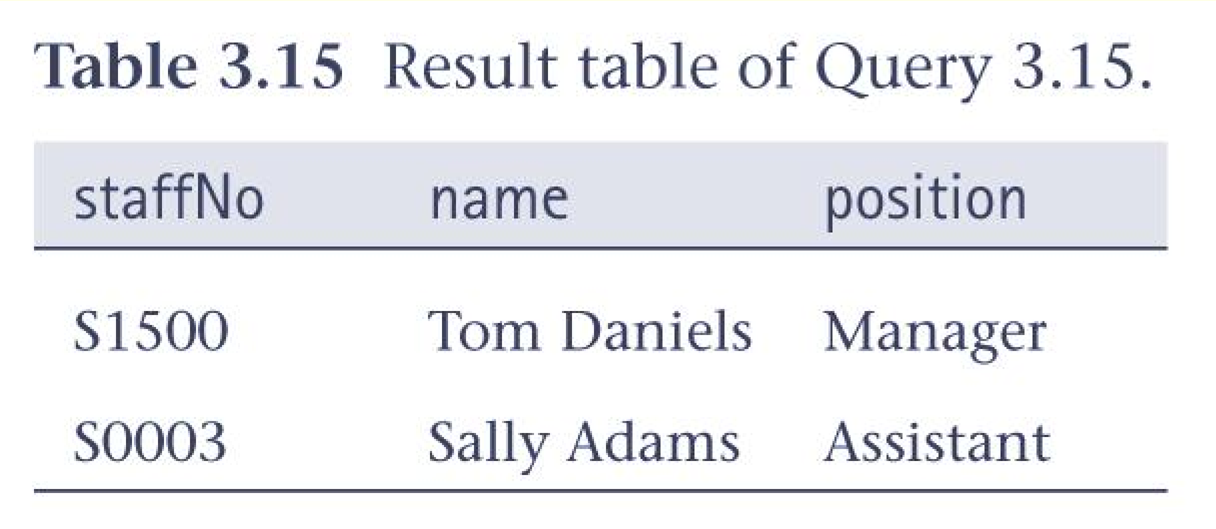


Table 3.15 Result table of Query 3.15.

| staffNo | name | position |
|---|---|---|
| S1500 | Tom Daniels | Manager |
| S0003 | Sally Adams | Assistant |

# 3.16 Subquery with Aggregate

List all staff whose salary is greater than the average salary.

```
SELECT staffNo, name, position
FROM Staff
WHERE salary >
          (SELECT AVG(salary)
           FROM Staff);
```

# 3.16 Subquery with Aggregate

- Cannot write ‘WHERE salary > AVG(salary)’
- Instead, use subquery to find average salary (41166.67), and then use outer SELECT to find those staff with salary greater than this:

```
SELECT staffNo, name, position
FROM Staff
WHERE salary > 41166.67;
```

# 3.16 Subquery with Aggregate

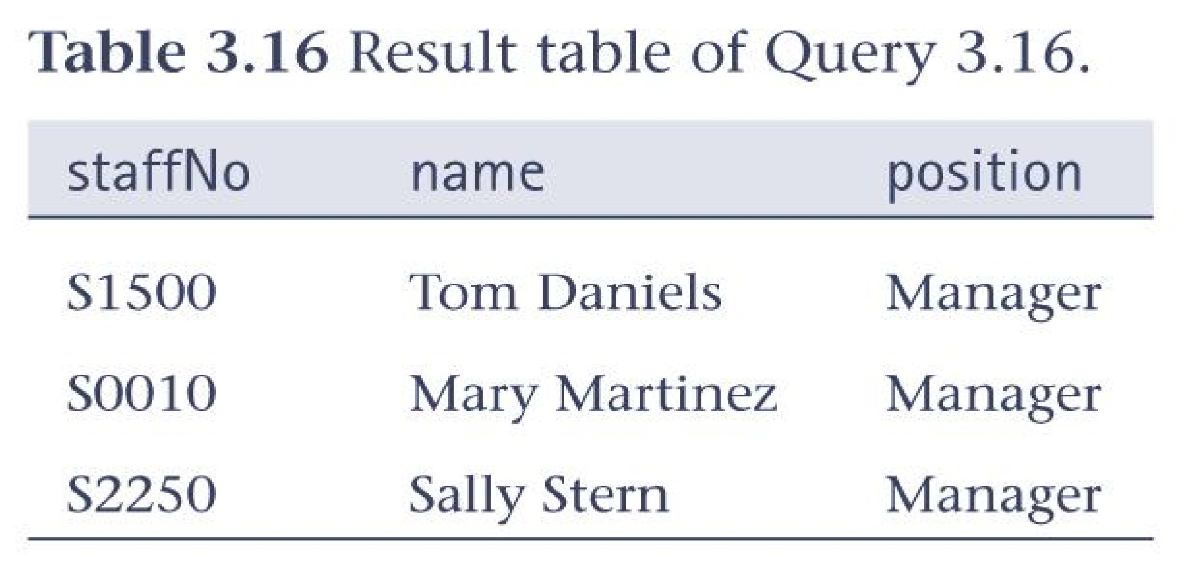


Table 3.16 Result table of Query 3.16.

| staffNo | name | position |
|---|---|---|
| S1500 | Tom Daniels | Manager |
| S0010 | Mary Martinez | Manager |
| S2250 | Sally Stern | Manager |

# UPDATE

```
UPDATE TableName
SET columnName1 = dataValue1
    [, columnName2 = dataValue2...]
[WHERE searchCondition]
```

- *TableName* can be name of a base table or an updatable view.
- SET clause specifies names of one or more columns that are to be updated.

# UPDATE

- WHERE clause is optional:
  - if omitted, named columns are updated for all rows in table;
  - if specified, only those rows that satisfy *searchCondition* are updated.
- New *dataValue(s)* must be compatible with data type for corresponding column.

# 3.20 UPDATE Rows in a Table

Modify the daily rental rate of videos in the ‘Thriller’ category by 10% .

```
UPDATE Video
SET dailyRental = dailyRental*1.1
WHERE category = ‘Thriller’;
```

# DELETE

```
DELETE FROM TableName
[WHERE searchCondition]
```

- *TableName* can be name of a base table or an updatable view.
- *searchCondition* is optional; if omitted, all rows are deleted from table. This does not delete table. If *searchCondition* specified, only those rows that satisfy condition are deleted.

# 3.21 DELETE Specific Rows

Delete rental videos for catalog number 634817 .

```
DELETE FROM VideoForRent
WHERE catalogNo = ‘634817’;
```

# Data Definition

- Two main SQL DDL statements:
  - CREATE TABLE - to create a new table.
  - CREATE VIEW - to create a new view.

# Defining a column

**columnName dataType [NOT NULL] [UNIQUE] [DEFAULT defaultOption]**

- **Supported data types of SQL are:**

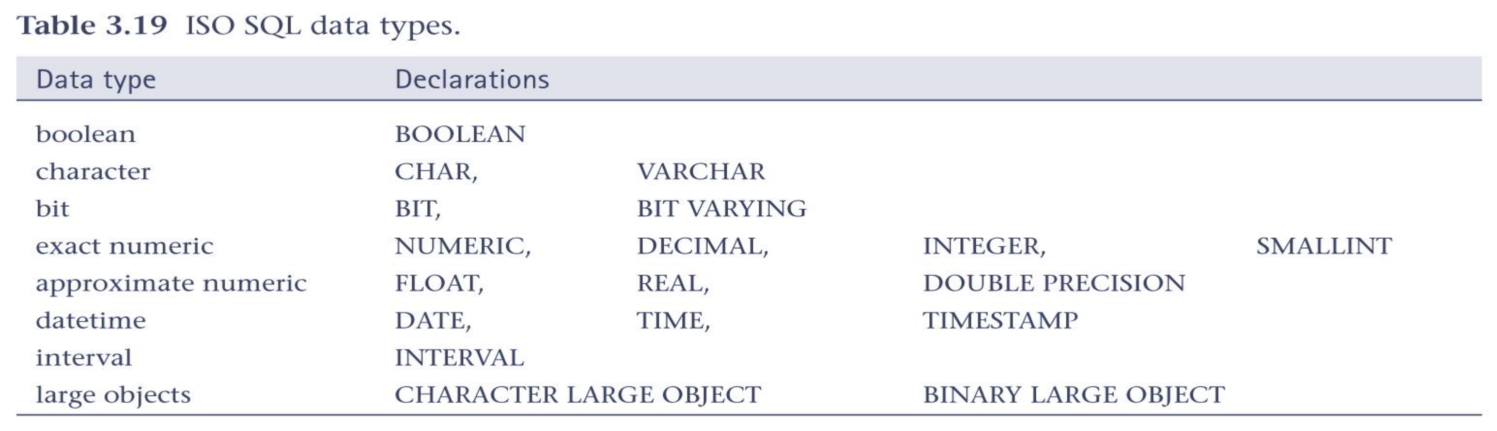


**Table 3.19** ISO SQL data types.

| Data type | Declarations | | | |
|---|---|---|---|---|
| boolean | BOOLEAN | | | |
| character | CHAR, | VARCHAR | | |
| bit | BIT, | BIT VARYING | | |
| exact numeric | NUMERIC, | DECIMAL, | INTEGER, | SMALLINT |
| approximate numeric | FLOAT, | REAL, | DOUBLE PRECISION | |
| datetime | DATE, | TIME, | TIMESTAMP | |
| interval | INTERVAL | | | |
| large objects | CHARACTER LARGE OBJECT | | BINARY LARGE OBJECT | |

# PRIMARY KEY and entity integrity

- Entity integrity supported by PRIMARY KEY clause.
- For example:

```
CONSTRAINT pk PRIMARY KEY (catalogNo)
CONSTRAINT pk1 PRIMARY KEY (catalogNo, actorNo)
```

# FOREIGN KEY and ref. integrity

- Use FOREIGN KEY clause to define any foreign keys in the table.
- SQL rejects any INSERT or UPDATE that attempts to create a FK value in child table without matching CK value in parent table.