Data Management and Visualization

Dr. Ashish Kumar Jha

Session 3

SQL Workshop

Agenda

SQL- Introduction

Database design method

ER Diagram to Database structure

Database Creation and Update

Keys

Loading values

Data Manipulation

Structured Query Language (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.

Structured Query Language (SQL)

- First and, so far, only standard database language to gain widespread acceptance.
- Huge investment from both vendors and users.
- Federal Information Processing Standard (FIPS).
- Used as the basis for other standards.
- ANSI and ISO standard is now the defining language for relational databases.

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.

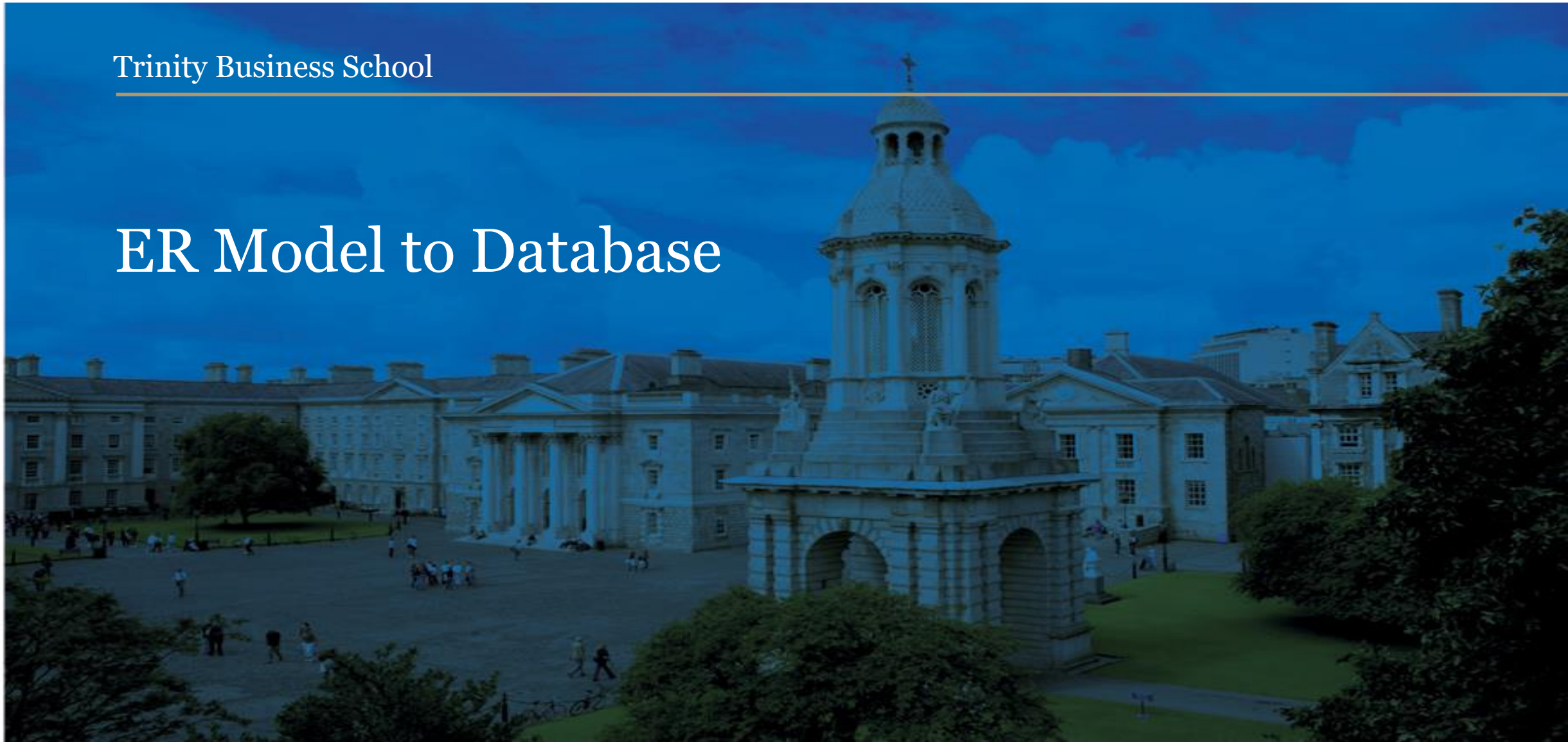
Must be portable.

Writing SQL Commands

- SQL statement consists of *reserved words* and *user-defined words*.
- Reserved words are a 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.

Literals

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


Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin
Trinity Business School
ER Model to Database

Conceptual database design

Step 1 Create ER model
- Step 1.1 Identify entities
- Step 1.2 Identify relationships
- Step 1.3 Identify and associate attributes with entities or relationships
- Step 1.4 Determine attribute domains
- Step 1.5 Determine candidate, primary, and alternate key attributes
- Step 1.6 Specialize/generalize entities (optional step)
- Step 1.7 Check the model for redundancy
- Step 1.8 Check that the model supports user transactions
- Step 1.9 Review the conceptual database design with users

Logical database design

Step 2 Map ER model to tables
- Step 2.1 Create tables
- Step 2.2 Check table structures using normalization
- Step 2.3 Check that the tables support user transactions
- Step 2.4 Check integrity constraints
- Step 2.5 Review the logical database design with users

Physical database design

Step 3 Translate the logical database design for target DBMS
- Step 3.1 Design base tables
- Step 3.2 Design representations of derived data
- Step 3.2 Design remaining integrity constraints

Step 4 Choose file organisations and indexes
- Step 4.1 Analyze transactions
- Step 4.2 Choose file organizations
- Step 4.3 Choose indexes

Step 5 Design user views

Step 6 Design security mechanisms

Step 7 Consider the introduction of controlled redundancy

Step 8 Monitor and tune the operational system

Logical database design (Map ER model to tables)

Step 2.1 Create tables

Step 2.2 Check table structures using normalization

Step 2.3 Check tables support user transactions

Step 2.4 Check integrity constraints

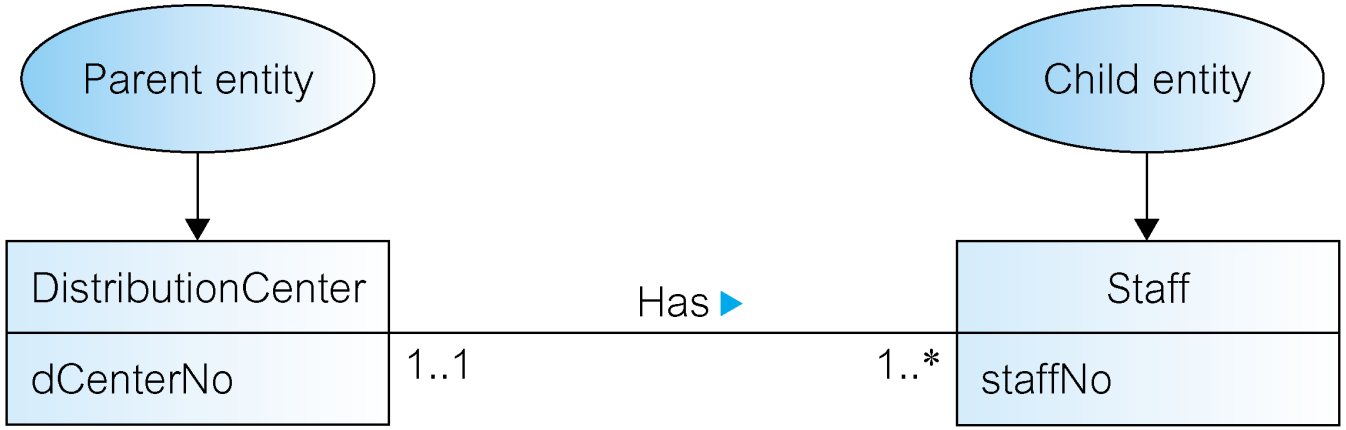
Step 2.5 Review logical database design with users

Initial table structures for *StayHome Online Rentals*

Actor (actorNo, actorName) **Primary key** actorNo	**DistributionCenter** (dCenterNo, dStreet, dCity, dState, dZipCode) **Primary key** dCenterNo **Alternate key** dZipCode
DVD (catalogNo, title, genre, rating) **Primary key** catalogNo	**DVDCopy** (DVDNo, available) **Primary key** DVDNo
DVDRental (deliveryNo, dateOut) **Primary key** deliveryNo	**Member** (memberNo, mFName, mLName, mStreet, mCity, mState, mZipCode, mEMail, mPword) **Primary key** memberNo **Alternate key** mEMail
MembershipType (mTypeNo, mTypeDesc, maxRentals, monthlyCharge) **Primary key** mTypeNo **Alternate key** mTypeDesc	**Staff** (staffNo, name, position, salary, eMail) **Primary key** staffNo **Alternate key** eMail
Wish (ranking) **Primary key** Not identified so far	

1:* relationship

(a)



(b)

`dCenterNo` is posted to represent the *Has* relationship

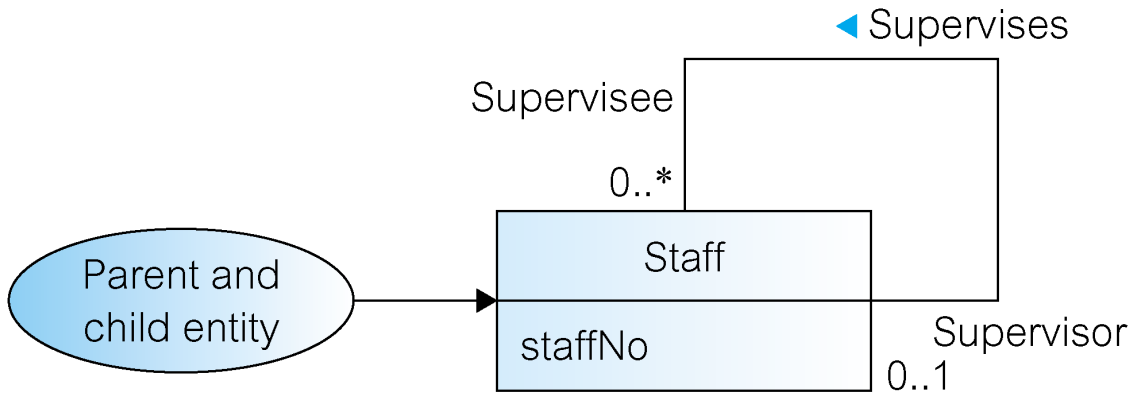
Foreign key

DistributionCenter `(dCenterNo, dStreet, dCity, dState, dZipCode)`
Primary key `dCenterNo`

Staff `(staffNo, name, position, salary, eMail, dCenterNo)`
Primary key `staffNo`
Foreign key `dCenterNo`

1:* recursive relationship

(a)



(b)

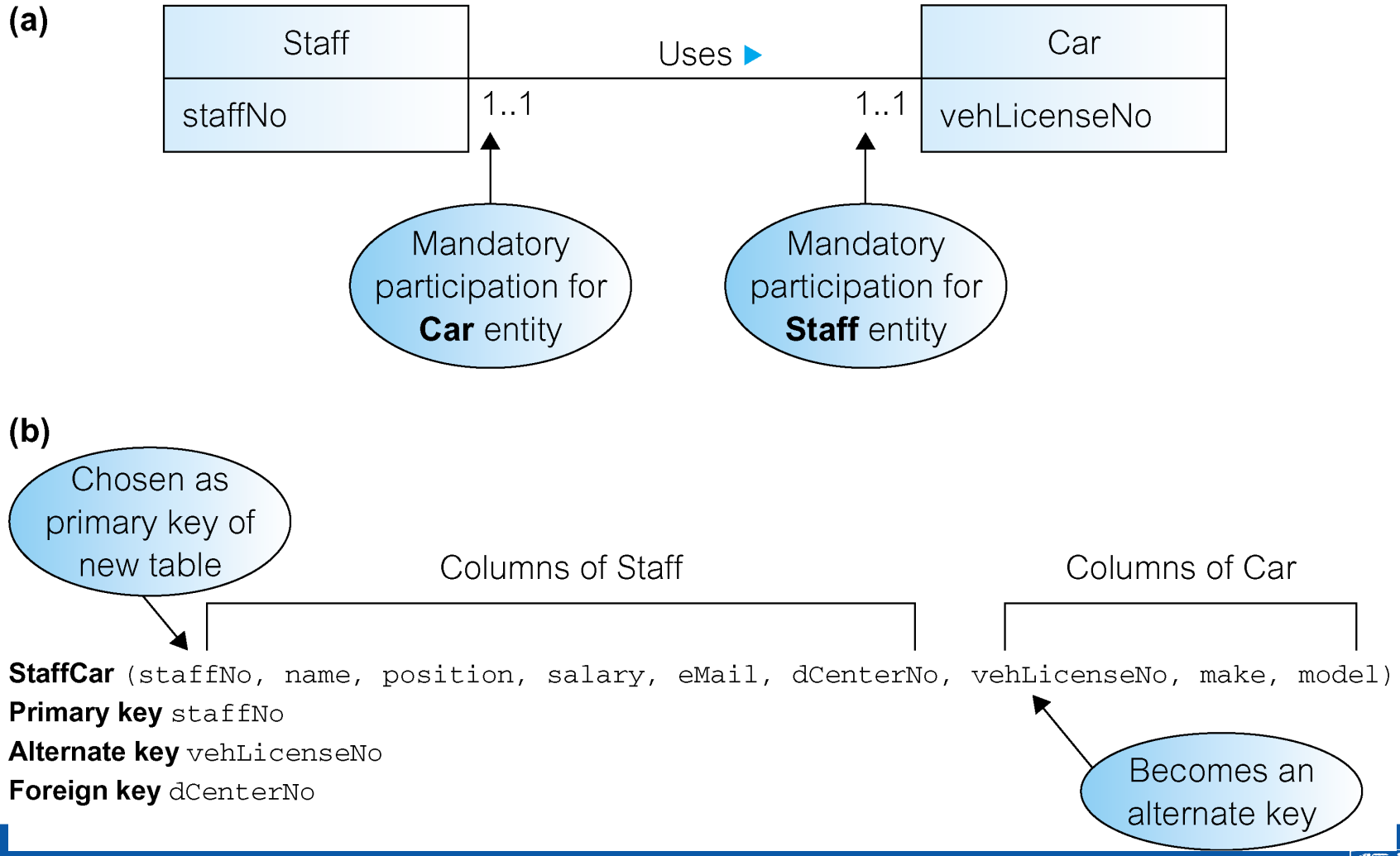
`staffNo` is posted to represent the Supervises relationship

Second copy of `staffNo` is renamed `supervisorStaffNo`

Staff (staffNo, name, position, salary, eMail, dCenterNo, supervisorStaffNo)
Primary key staffNo
Foreign key dCenterNo
Foreign key supervisorStaffNo

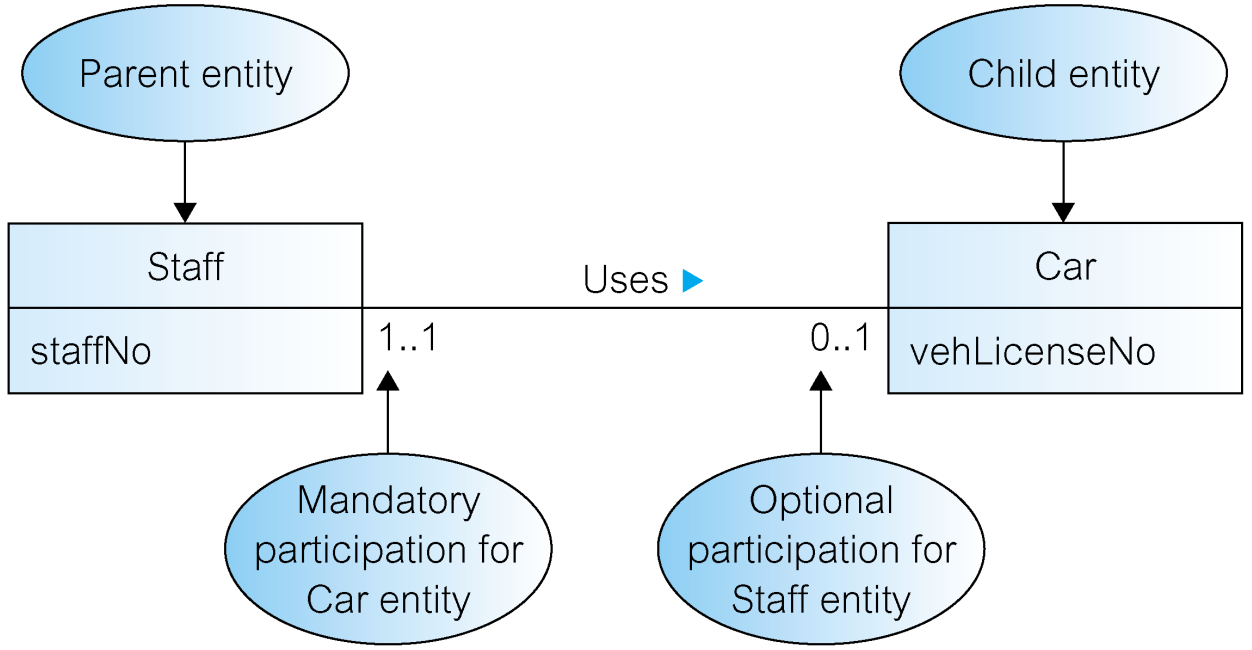
dCenterNo represents Branch *Has* Staff relationship

1:1 relationship mandatory both entities



1:1 relationship mandatory one entity only

(a)



(b)

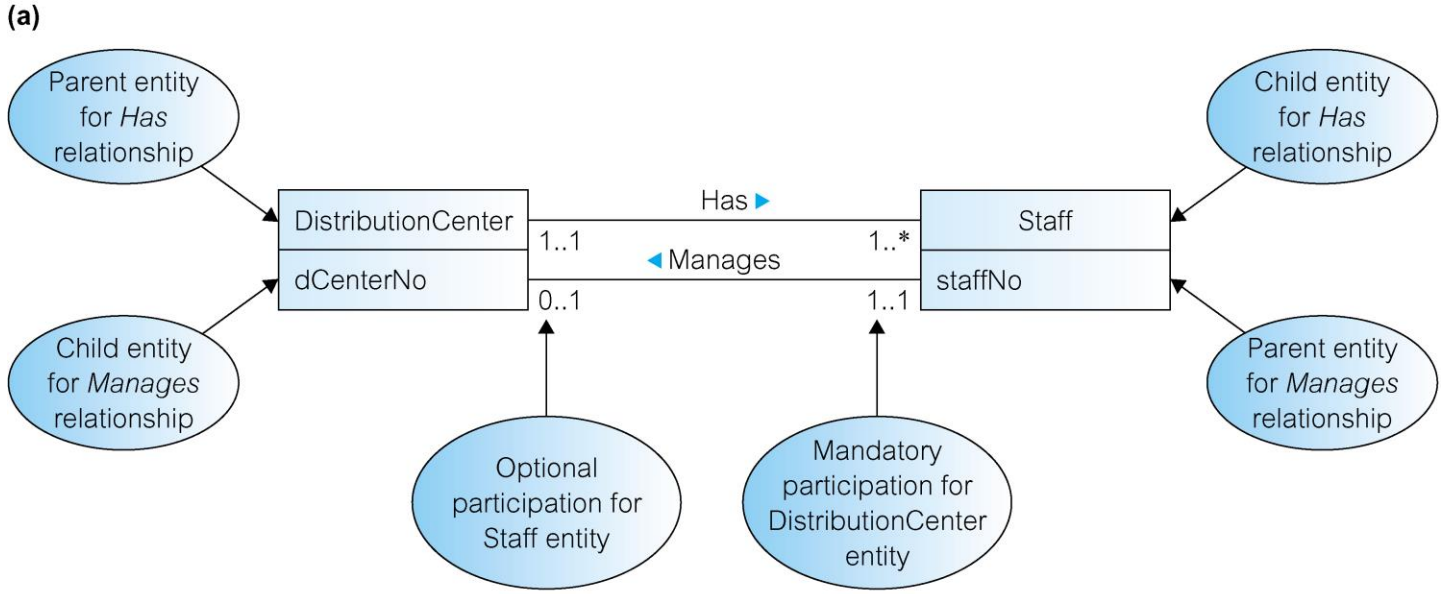
`staffNo` is posted to represent the *Uses* relationship

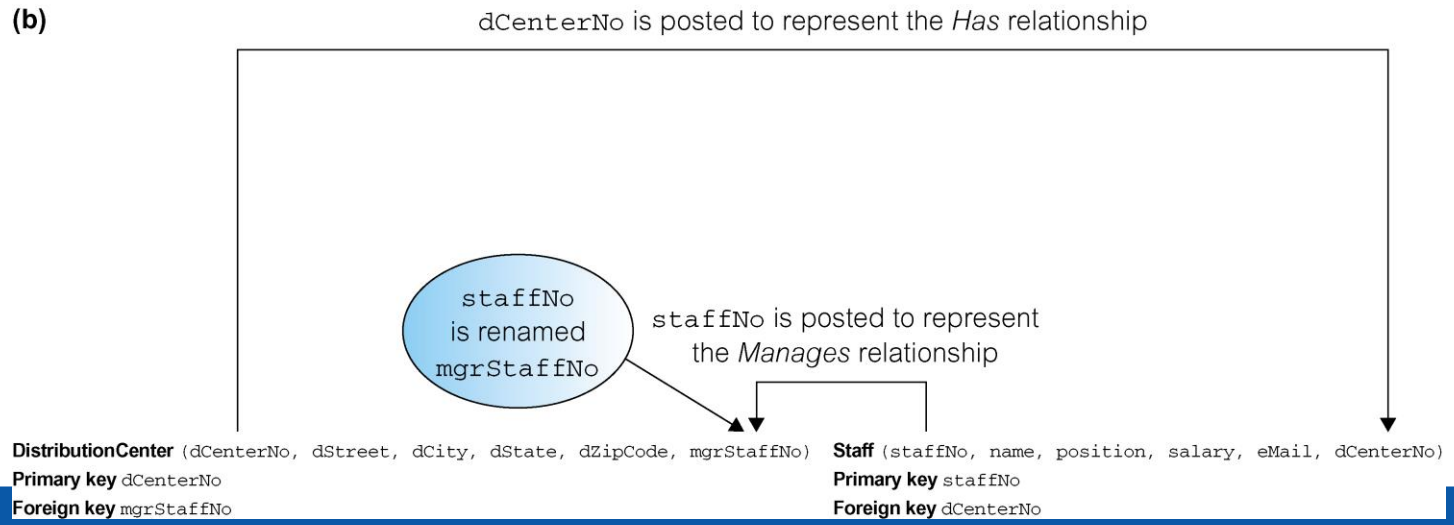
Foreign key

Staff (`staffNo, name, position, salary, eMail, dCenterNo`)
Primary key `staffNo`
Foreign key `dCenterNo`

Car (`vehLicenseNo make, model, staffNo`)
Primary key `vehLicenseNo`
Alternate key `staffNo`

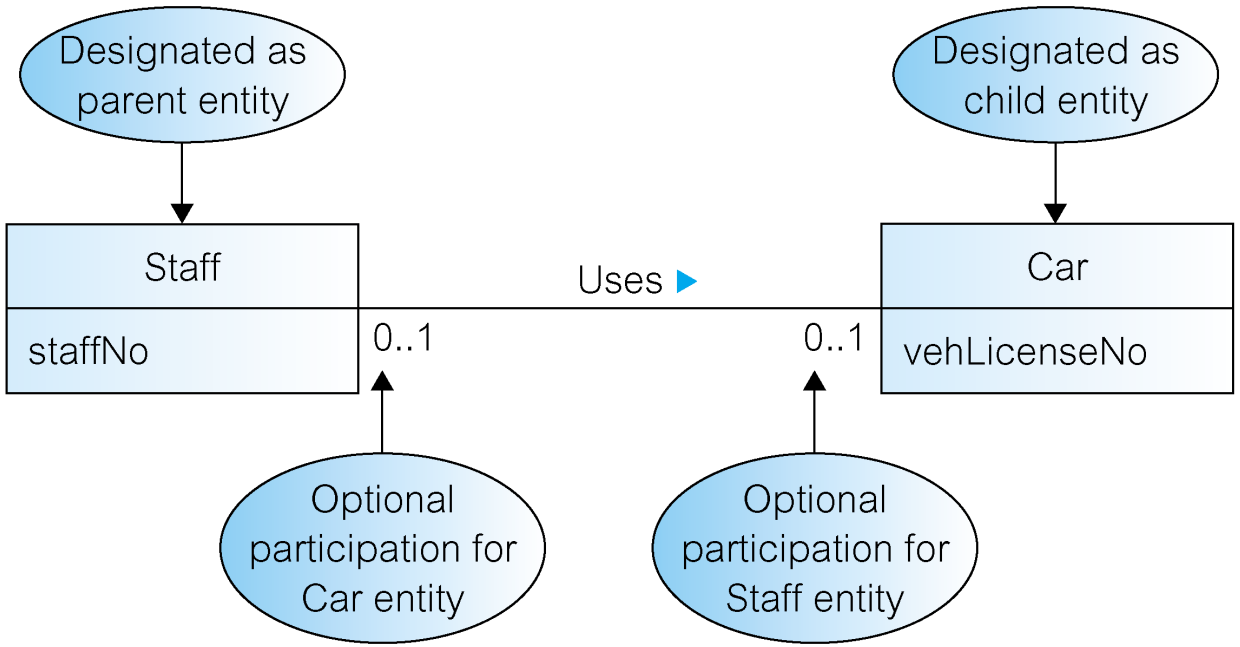
1:1 relationship mandatory one entity only





1:1 relationship optional both entities

(a)



(b)

`staffNo` is posted to represent the *Uses* relationship

Foreign key

Staff (`staffNo, name, position, salary, eMail, dCenterNo`)
Primary key `staffNo`
Foreign key `dCenterNo`

Car (`vehLicenseNo, make, model, staffNo`)
Primary key `vehLicenseNo`
Alternate key `staffNo`
Foreign key `staffNo`

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Trinity Business School

Database Creation

MYSQL

Workbench introduction

Commands

```
CREATE DATABASE test_new;

USE test_new;

CREATE TABLE limbs (thing VARCHAR(20), legs INT, arms INT);
```

```
INSERT INTO limbs (thing,legs,arms) VALUES('human',2,2);

INSERT INTO limbs (thing,legs,arms) VALUES('insect',6,0);

INSERT INTO limbs (thing,legs,arms) VALUES('squid',0,10);

INSERT INTO limbs (thing,legs,arms) VALUES('fish',0,0);
```

Autoincrement

```
CREATE TABLE Persons (
    Personid int NOT NULL AUTO_INCREMENT,
    FirstName varchar(255),
    LastName varchar(255) NOT NULL,
    Age int,
    PRIMARY KEY (Personid)
);
```

```
INSERT INTO Persons (FirstName,LastName)
VALUES (‘Ashish’,’Jha');
```

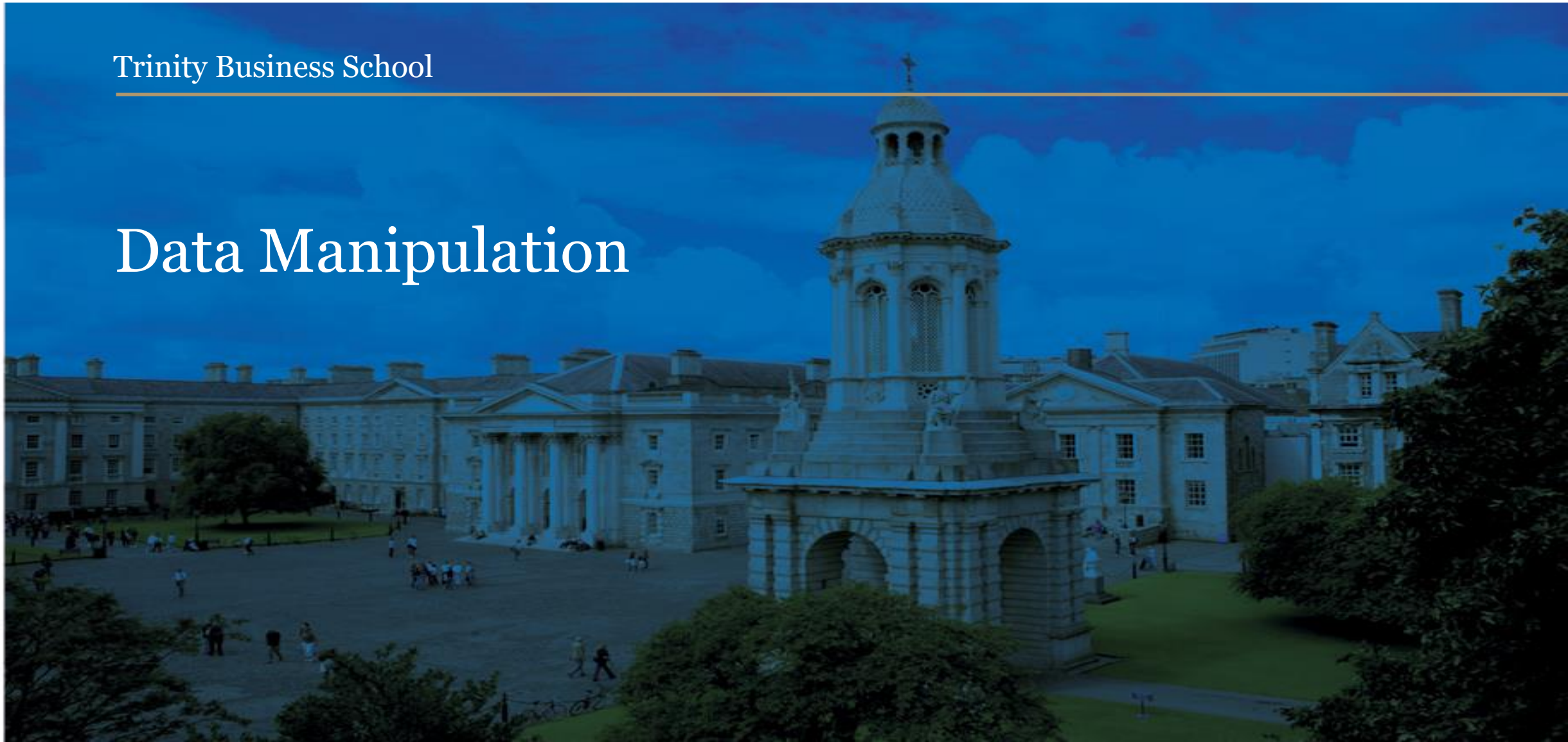
Declaring Keys

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Trinity Business School

Data Manipulation



Data Manipulation – Main Statements

SELECT to query data in the database

INSERT to insert data into a table

UPDATE to update data in a table

DELETE to delete data from a table

Simple Queries - SELECT Statement

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

SELECT Statement Key Points

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

Order of the clauses cannot be changed.

Only SELECT and FROM are mandatory.

Query 3.1 All columns, all rows

List full details of all DVDs.

```
SELECT catalogNo, title, genre, rating
FROM DVD;
```

Can use * as an abbreviation for 'all columns':

```
SELECT *
FROM DVD;
```

Query 3.1 All columns, all rows

Table 3.1 Result table for Query 3.1

catalogNo	title	genre	rating
207132	Casino Royale	Action	PG-13
902355	Harry Potter and the GOF	Children	PG
330553	Lord of the Rings III	Action	PG-13
781132	Shrek 2	Children	PG
445624	Mission Impossible III	Action	PG-13
634817	War of the Worlds	Sci-Fi	PG-13

Query 3.2 Specific columns, all rows

List the catalog number, title and genre of all DVDs.

```
SELECT catalogNo, title, genre
FROM DVD;
```

Table 3.2 Specific Columns, All Rows

Table 3.2 Result table for Query 3.2

catalogNo	title	genre
207132	Casino Royale	Action
902355	Harry Potter and the GOF	Children
330553	Lord of the Rings III	Action
781132	Shrek 2	Children
445624	Mission Impossible III	Action
634817	War of the Worlds	Sci-Fi

Query 3.3 Use of DISTINCT

List all DVD genres.

SELECT genres

FROM DVD;

SELECT DISTINCT genres

FROM DVD;

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

genre
Action
Children
Action
Children
Action
Sci-Fi

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

genre
Action
Children
Sci-Fi

Calculated Fields

List the monthly salary for all staff, showing the staff number, name, position and monthly salary.

SELECT staffNo, name, position, salary/12

FROM Staff;

Table 3.4 Result table of Query 3.4

staffNo	name	position	col4
S1500	Tom Daniels	Manager	4000
S0003	Sally Adams	Assistant	2500
S0010	Mary Martinez	Manager	4250
S3250	Robert Chin	Assistant	2750
S2250	Sally Stern	Manager	4000
S0415	Art Peters	Manager	3500

Row Selection (WHERE clause)

Five basic search conditions include:

- Comparison : compare the value of one expression to the value of another.
- Range: test whether value falls within a specified range.
- Set membership: test whether the value of an expression equals one of a set of values.
- Pattern match: test whether a string matches a specified pattern.
- Null: test whether a column has a unknown value.

Query 3.5 Comparison Search Condition

List all staff with a salary greater than $40,000.

```
SELECT staffNo, name, position, salary
FROM Staff
WHERE salary > 40000;
```

Table 3.5 Result table for Query 3.5

staffNo	name	position	salary
S1500	Tom Daniels	Manager	48000
S0010	Mary Martinez	Manager	51000
S2250	Sally Stern	Manager	48000
S0415	Art Peters	Manager	42000

Query 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 >= 45000 **AND** salary <= 50000;

Here we use the logical operator AND in the WHERE clause to find the rows in the Staff table where the value in the salary column is between $45 000 and $50 000

Result 3.6 Range Search Condition

Table 3.6 Result table for Query 3.6

staffNo	name	position	salary
S1500	Tom Daniels	Manager	48000
S2250	Sally Stern	Manager	48000

Query 3.7 Set Membership

List all DVDs in the Sci-Fi or Children genres.

SELECT catalogNo, title, genres
FROM DVD
WHERE genre='Sci-Fi' **OR** genre='Children';

Table 3.7 Result table for Query 3.7

catalogNo	title	genre
902355	Harry Potter and the GOF	Children
781132	Shrek 2	Children
634817	War of the Worlds	Sci-Fi

Query 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, genre
FROM DVD
WHERE genre IN ('Sci-Fi', 'Children');
```

IN is more efficient when set contains many values.

Query 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

Query 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.

Query 3.9 NULL Search Condition

List the rentals that have no return date specified.

SELECT deliveryNo, DVDNo

FROM DVDRental

WHERE dateReturn **IS NULL**;

Have to test for null explicitly using special keyword IS NULL (IS NOT NULL).

Table 3.9 Result table for Query 3.9

deliveryNo	DVDNo
R66818964	17864331

Sorting Results (ORDER BY)

List all DVDs, sorted in descending order of genre.

```
SELECT *
FROM DVD
ORDER BY genre DESC;
```

Table 3.10(a) Result table for Query 3.10 with ordering on `genre`

catalogNo	title	genre	rating
634817	War of the Worlds	Sci-Fi	PG-13
902355	Harry Potter and the GOF	Children	PG
781132	Shrek 2	Children	PG
207132	Casino Royale	Action	PG-13
330553	Lord of the Rings III	Action	PG-13
445624	Mission Impossible III	Action	PG-13

Table 3.10 Single Column Ordering

We can add a minor ordering clause to sort the same genres on catalogNo:

ORDER BY genre **DESC**, catalogNo ASC;

Table 3.10(b) Result table for Query 3.10 with ordering on `genre` and `catalogNo`

catalogNo	title	genre	rating
634817	War of the Worlds	Sci-Fi	PG-13
781132	Shrek 2	Children	PG
902355	Harry Potter and the GOF	Children	PG
207132	Casino Royale	Action	PG-13
330553	Lord of the Rings III	Action	PG-13
445624	Mission Impossible III	Action	PG-13

Using the SQL Aggregate Functions

ISO SQL standard 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.

Using the SQL Aggregate Functions

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.

Using the SQL Aggregate Functions

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.

Using the SQL Aggregate Functions

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;
```

Query 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
WH
```

Table 3.11 Result table of Query 3.11

totalStaff	totalSalary
4	189000

Query 3.12 Use of MIN, MAX, 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	51000	42000

Grouping Results

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.

Grouping Results

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.

Query 3.13 Use of GROUP BY

Find number of staff working in each distribution center and the sum of their salaries.

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

Table 3.13 Result table for Query 3.13

dCenterNo	totalStaff	totalSalary
D001	2	78000
D002	2	84000
D003	1	42000
D004	1	48000

Grouping Results by 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.

Query 3.1.4 Use of HAVING

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

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

Results 3.14 Use of HAVING

Table 3.14 Result table of Query 3.14

dCenterNo	totalStaff	totalSalary
D001	2	78000
D002	2	84000

Subselects

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.

Query 3.15 Using a subquery

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

```
SELECT staffNo, name, position
FROM Staff
WHERE dCenterNo=(SELECT dCenterNo
                 FROM DistributionCenter
                 WHERE dStreet=‘8 Jefferson Way’);
```

Using a Subquery

Inner SELECT finds distribution center number for distribution center at ‘8 Jefferson Way’ (‘B001’).

Outer SELECT then retrieves details of all staff who work at this center.

Outer SELECT then becomes:

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

Results 3.15 Subquery

Table 3.15 Result table of Query 3.15

staffNo	name	position
S1500	Tom Daniels	Manager
S0003	Sally Adams	Assistant

Query 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);
```

Query 3.16 Subquery with Aggregate

Cannot write 'WHERE salary > AVG(salary)'

Instead, use subquery to find average salary (42000), and then use outer SELECT to find those staff with salary greater than this average:

```
SELECT staffNo, name, position
FROM Staff
WHERE salary > 42000;
```

Query 3.16 Result 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

Subquery Rules: Key points

ORDER BY clause may not be used in a subquery (although it may be used in outermost SELECT).

Subquery SELECT list must consist of a single column name or expression, except for subqueries that use EXISTS.

By default, column names refer to table name in FROM clause of subquery. Can refer to a table in FROM using an *alias*.

Subquery Rules: Key points

When subquery is an operand in a comparison, subquery must appear on right-hand side.

A subquery may not be used as an operand in an expression.

Multi-Table Queries

Can use subqueries provided result columns come from same table.

If result columns come from more than one table must use a join.

To perform join, include more than one table in FROM clause.

Use comma as separator with typically a WHERE to specify join column(s).

Multi-Table Queries

Also possible to use an alias for a table named in FROM clause.

Alias is separated from table name with a space.

Alias can be used to qualify column names when there is ambiguity.

Query 3.17 Simple Join

List all actors and the characters they have played in DVDs.

```
SELECT a.actorNo, actorName, character
FROM Actor a, DVDActor da
WHERE a.actorNo = da.actorno:
```

Simple Join

Only those rows from both tables with identical values in the actorNo columns (a. actorNo = da. actorNo) included in result.

Table 3.17 Result table of Query 3.17

actorNo	actorName	character
A1002	Judi Dench	M
A3006	Elijah Wood	Frodo Baggins
A2019	Tom Cruise	Ethan Hunt
A2019	Tom Cruise	Ray Ferrier
A7525	Ian McKellen	Gandalf
A4343	Mike Myers	Shrek
A8401	Daniel Radcliffe	Harry Potter

Alternative JOIN Constructs

Alternative ways to specify joins:

```
FROM Actor a JOIN DVDActor da ON a.actorNo = da.actorNo;
```

```
FROM Actor JOIN DVDActor USING actorNo
```

```
FROM Actor NATURAL JOIN DVDActor
```

Query 3.18 Three Table Join

List all actors and the characters they have played in DVDs, along with the DVD's title.

```
SELECT  a. actorNo, actorName, character, title
FROM  Actor  a,  DVDActor  da,  DVD  d
WHERE a. actorNo  =  da.actorNo AND
          da. catalogNo  =  d.catalogNo;
```

Query 3.18 Three Table Join

Table 3.18 Result table of Query 3.18

actorNo	actorName	character	title
A1002	Judi Dench	M	Casino Royale
A3006	Elijah Wood	Frodo Baggins	Lord of the Rings III
A2019	Tom Cruise	Ethan Hunt	Mission Impossible III
A2019	Tom Cruise	Ray Ferrier	War of the Worlds
A7525	Ian McKellen	Gandalf	Lord of the Rings III
A4343	Mike Myers	Shrek	Shrek 2
A8401	Daniel Radcliffe	Harry Potter	Harry Potter and the GOF

EXISTS and NOT EXISTS

The keywords EXISTS and NOT EXISTS are designed for use only with sub-queries. They produce a simple true/false result.

EXISTS is true if and only if there exists at least one row in the result table returned by the subquery; it is false if the subquery returns an empty table.

For simplicity, it is common for subqueries following one of these keywords to be of the form:

(SELECT * FROM……)

Query 3.19 Query using EXISTS

Find all staff who work in the Washington distribution center.

```
SELECT staffNo, name, position
FROM Staff   s
WHERE EXISTS (SELECT *
                    FROM DistributionCenter  d
                    WHERE s.dCenterNo = d. dCenterNo
                              AND dState = 'WA');
```

Query 3.19 Query Using EXISTS

Table 3.19 Result table of Query 3.19

staffNo	name	position
S0010	Mary Martinez	Manager
S3250	Robert Chin	Assistant

INSERT – Add new row(s) to 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 – Add new row(s) to 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.

UPDATE existing data in table

The format of the UPDATE statement is:

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 existing data in table

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.

DELETE rows of data from a table

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.**