Edith Cowan University

School of Science

CSI6207
Systems Analysis and Database Design

SQL Basics

Objectives

- Entity Relationship (ER) Diagram
  - – Multiple and Self-referencing relationships

- Introduction to SQL
- Sample database (used in lectures and labs)
- Simple SELECT statements
- SQL Server & SQL Server Management Studio

Multiple Relationships

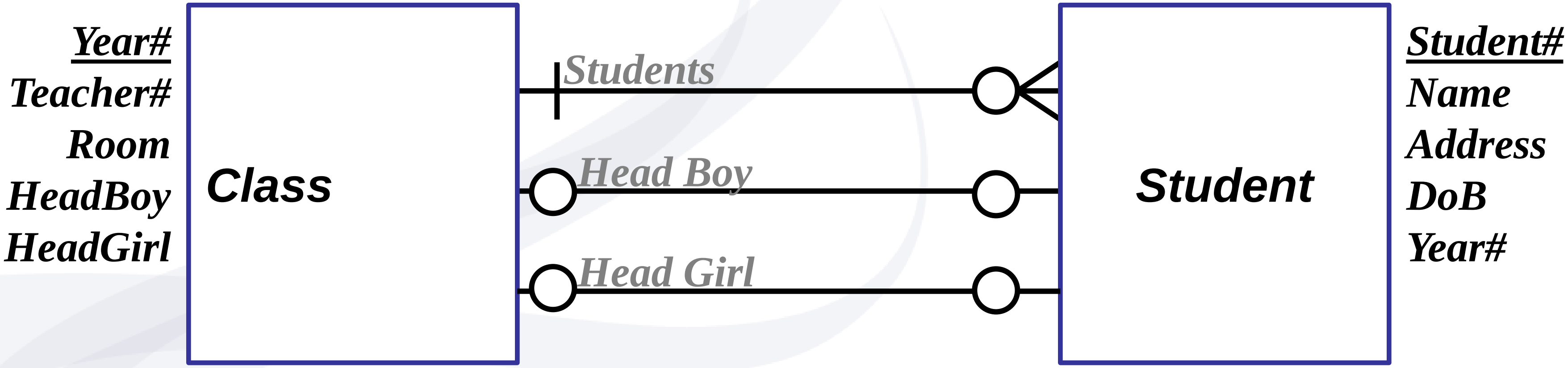
- In the examples in the previous lecture, there was only *one relationship between two given entities*
  - This is NOT a rule

- Multiple relationships may occur when *two entities are linked in more than one way*

Multiple Relationships

- Imagine a *primary school* database:
  - Each Class (e.g. “Year 6”) has multiple students
  - Each Class has a Head Boy
  - Each Class has a Head Girl

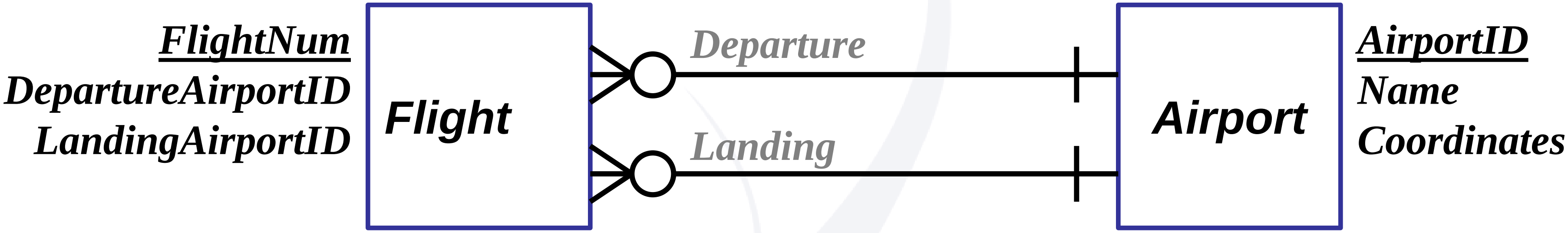
  - Each year *may have* a head boy and a head girl, and it is not required for a student to be a head boy or girl



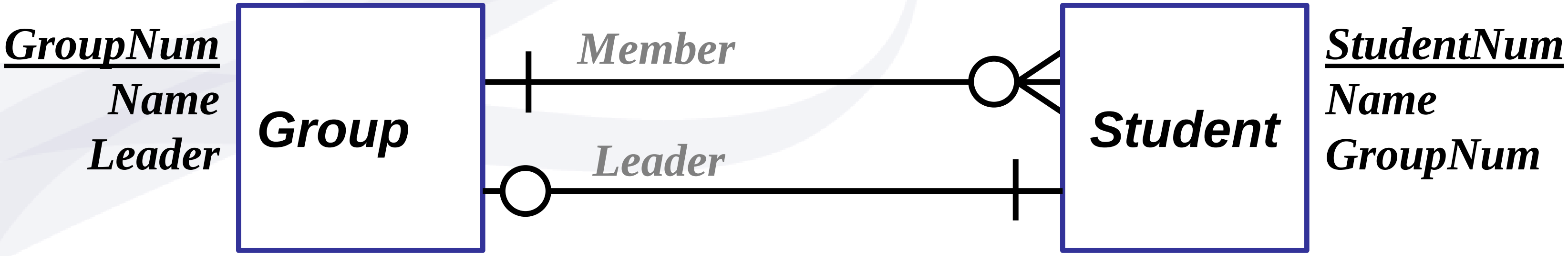
- When multiple relationships exist, it is important to give FKs meaningful names, and consider naming the relationships

Multiple Relationships

- Other examples of multiple relationship scenarios:
  - "A flight departs from one airport and lands at another"

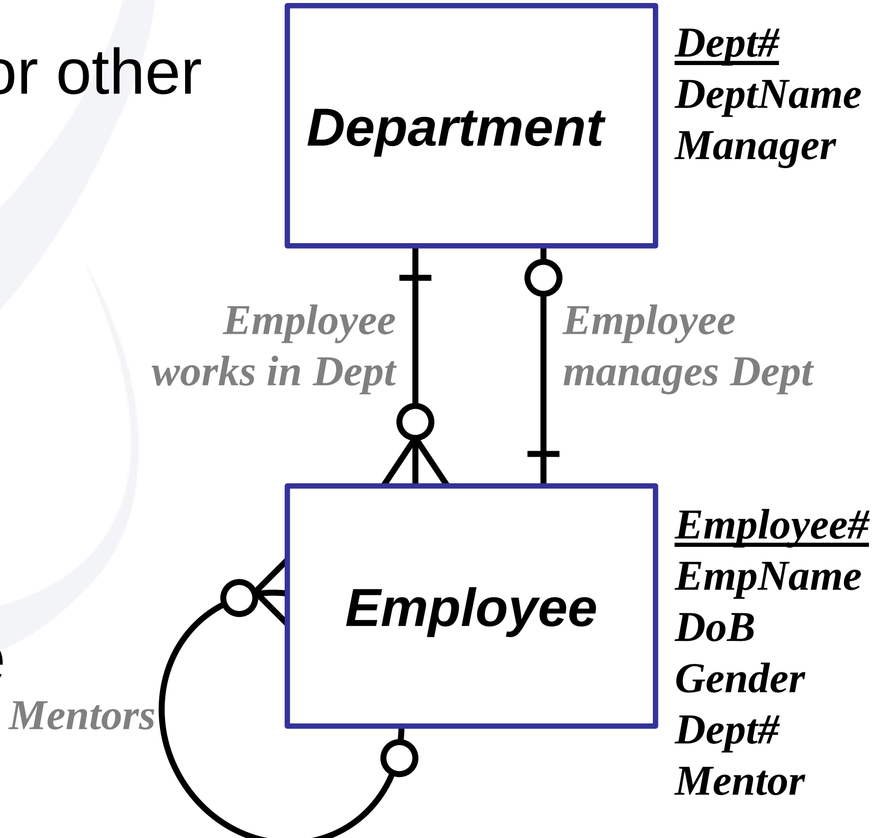


  - "Groups can have many students and must have a leader"

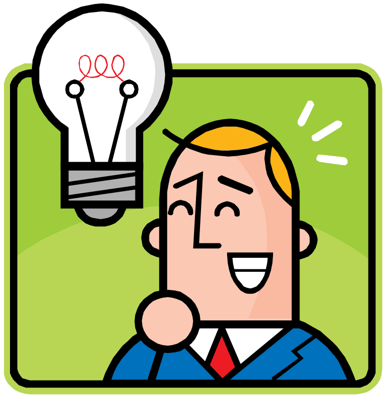


Self-Referencing Relationships

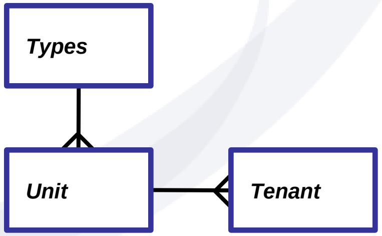
- A company database, with departments and employees
  - One department may have multiple employees
  - Each department also has a department manager
  - Employees may also mentor other employees in a 1:M way
- This is a self-referencing relationship
- Employee has a recursive relationship with itself



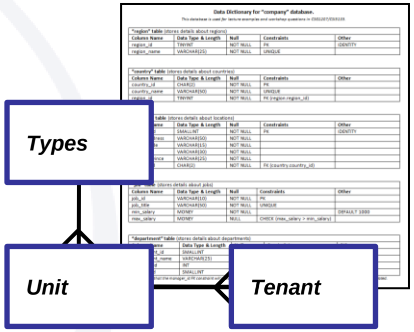
From Concept to Database



Model of system in client's mind



Physical E-R model of client's model
(in 3NF or higher)



Data Dictionary for "company" database.

This database is used for lecture examples and workshop questions in CSG1207/CSI5135.

"region" table (stores details about regions)

Column Name	Data Type & Length	Null	Constraints	Other
region_id	TINYINT	NOT NULL	PK	IDENTITY
region_name	VARCHAR(25)	NOT NULL	UNIQUE	

"country" table (stores details about countries)

Column Name	Data Type & Length	Null	Constraints	Other
country_id	CHAR(2)	NOT NULL	PK	
country_name	VARCHAR(50)	NOT NULL	UNIQUE	
region_id	TINYINT	NOT NULL	FK (region.region_id)	

[illegible] **table** (stores details about locations)

[illegible]ame	Data Type & Length	Null	Constraints	Other
[illegible]d	SMALLINT	NOT NULL	PK	IDENTITY
[illegible]dress	VARCHAR(50)	NOT NULL		
[illegible]de	VARCHAR(15)	NOT NULL		
[illegible]	VARCHAR(30)	NOT NULL		
[illegible]vince	VARCHAR(25)	NOT NULL		
[illegible]d	CHAR(2)	NOT NULL	FK (country.country_id)	

[illegible] (stores details about jobs)

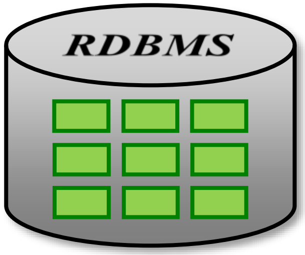
Column Name	Data Type & Length	Null	Constraints	Other
job_id	VARCHAR(10)	NOT NULL	PK	
job_title	VARCHAR(50)	NOT NULL	UNIQUE	
min_salary	MONEY	NOT NULL		DEFAULT 1000
max_salary	MONEY	NULL	CHECK (max_salary > min_salary)	

"department" table (stores details about departments)

[illegible]ame	Data Type & Length	[illegible]	[illegible]	[illegible]
[illegible]nt_id	SMALLINT	[illegible]	[illegible]	[illegible]
[illegible]nt_name	VARCHAR(25)	[illegible]	[illegible]	[illegible]
[illegible]d	INT	[illegible]	[illegible]	[illegible]
[illegible]d	SMALLINT	[illegible]	[illegible]	[illegible]

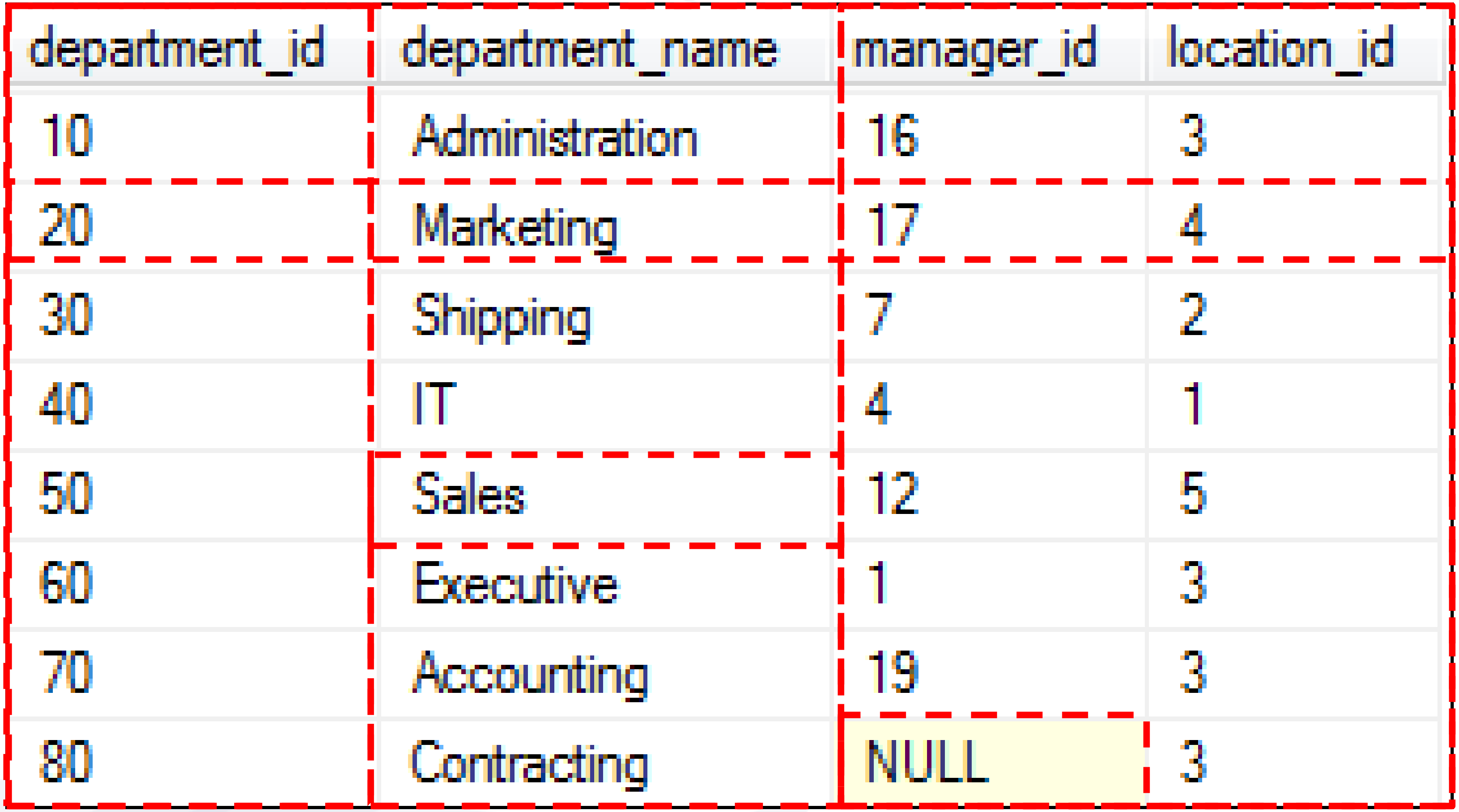
[illegible] that the manager_id FK constraint wi[illegible] [illegible]ded.

Full database design
(including data dictionary)



Tables on disk in a RDBMS

Relational Database Terminology Recap



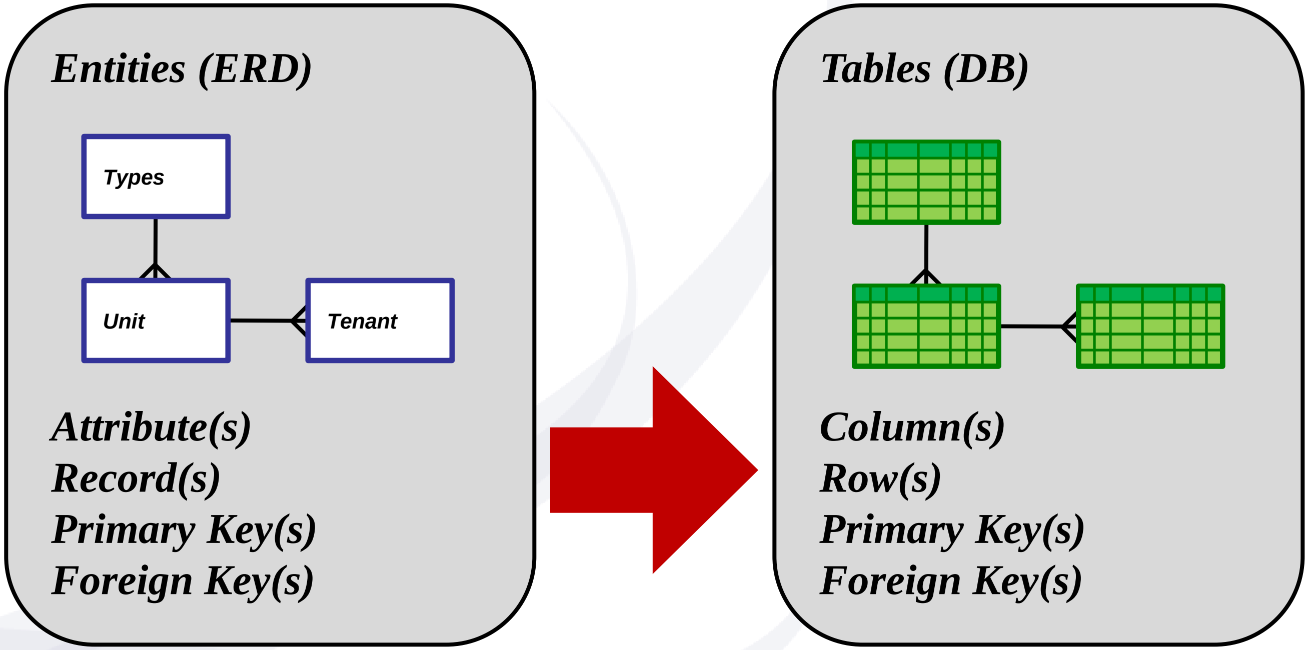
department_id	department_name	manager_id	location_id
10	Administration	16	3
20	Marketing	17	4
30	Shipping	7	2
40	IT	4	1
50	Sales	12	5
60	Executive	1	3
70	Accounting	19	3
80	Contracting	NULL	3

Table *(Relation)*. **Row (Record).** **Column (Attribute).**

Primary Key. **Foreign Keys.** **Field.** **Null** *(No Value)*.

ERD to Database Tables

- Each entity in a physical ERD *translates directly* to a table



Entities (ERD)

Attribute(s)
Record(s)
Primary Key(s)
Foreign Key(s)

Tables (DB)

Column(s)
Row(s)
Primary Key(s)
Foreign Key(s)

Database Table Creation

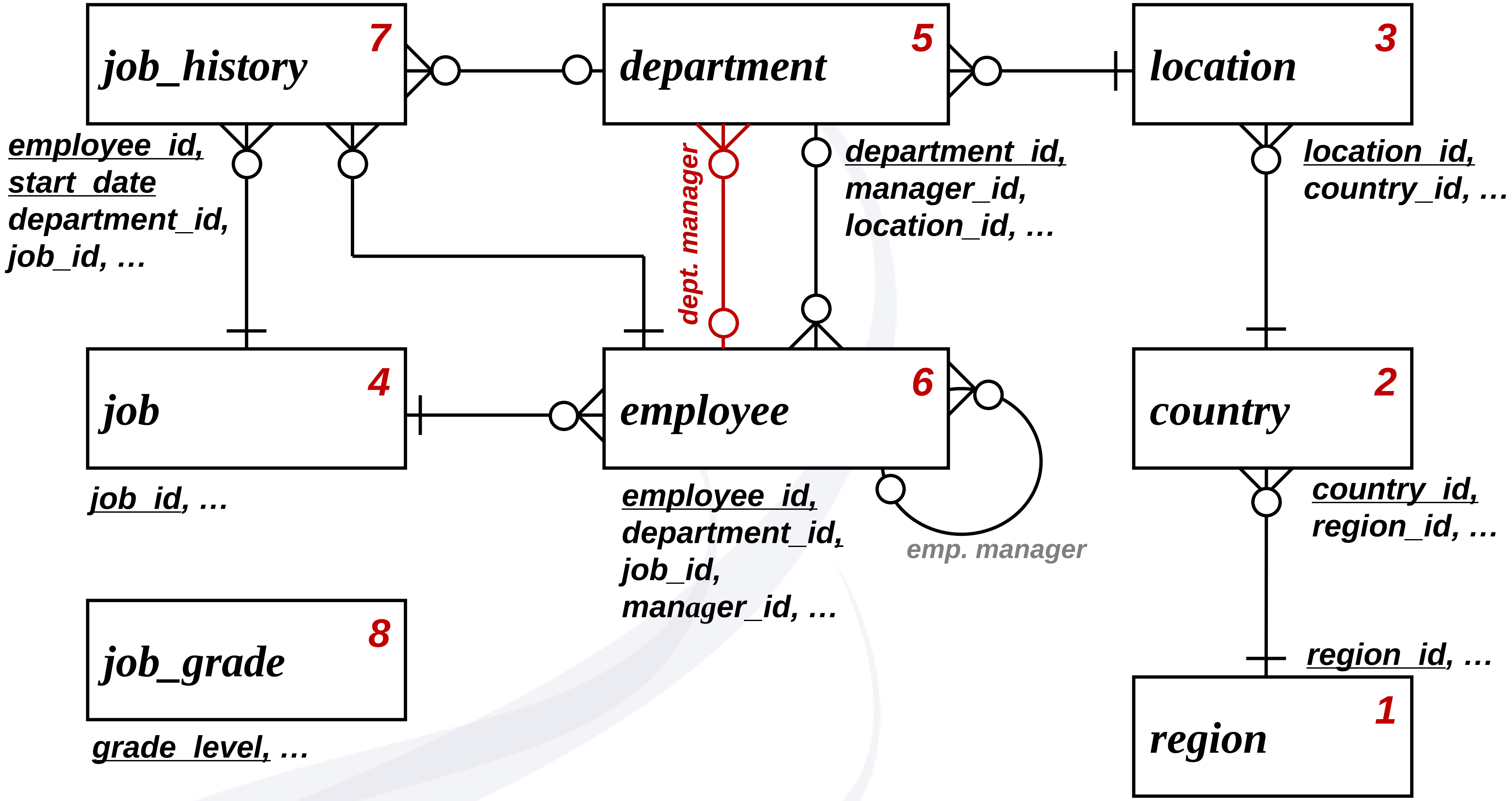
- Once a physical model of a normalised system has been created, it's time to *implement it as an actual database*

- The first step is to **create the tables** of your database

- Each entity in your physical model will become a table

- The **order** of table creation is important, in order to ensure the existence of primary and foreign keys in relationships
  - *You cannot have a foreign key column that refers to a table that has not yet been created*

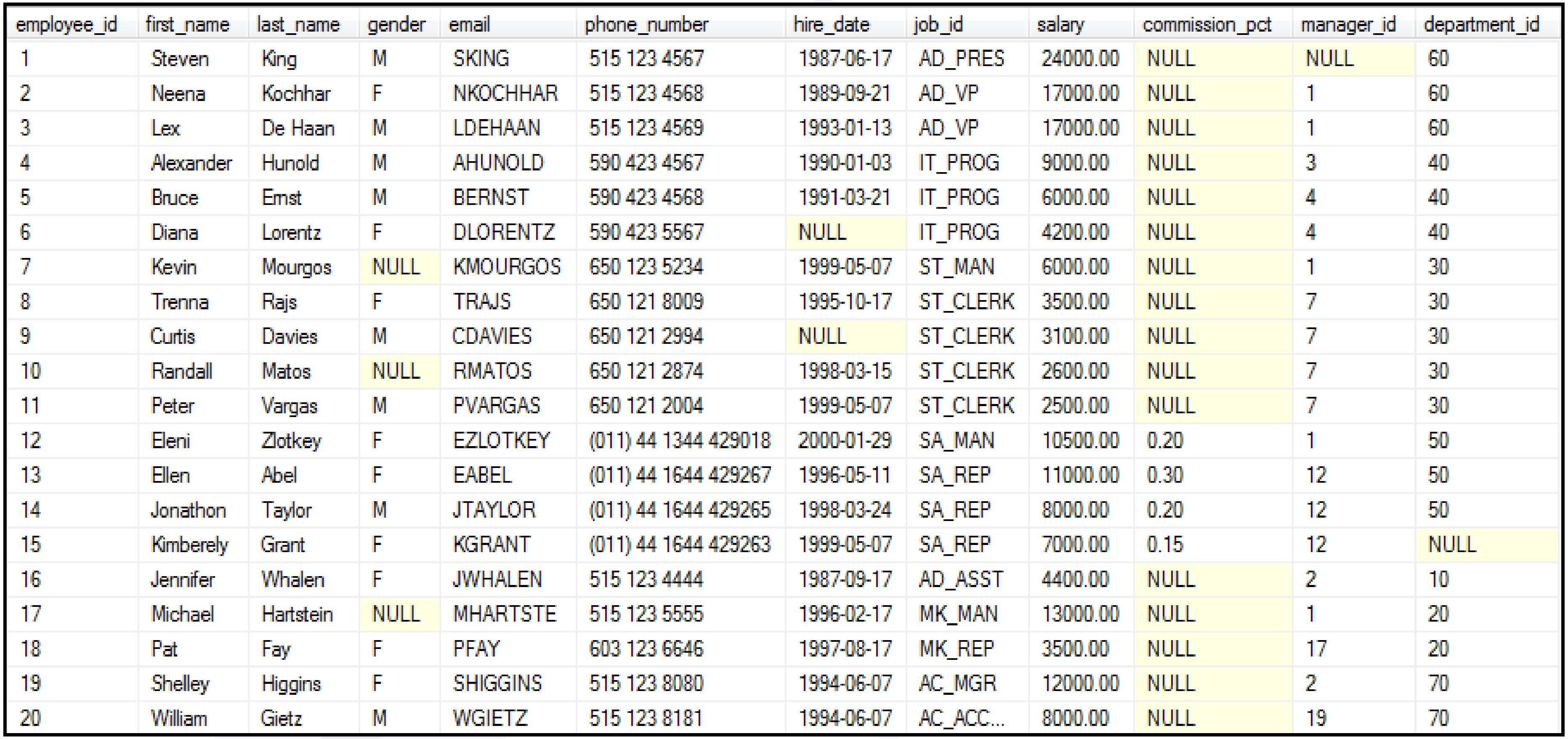
Sample Database – “Company”



- job_grade has no FKs – creation order not important
- The relationship in red is added *after creation and population* to prevent issues with circular references

The Company Database

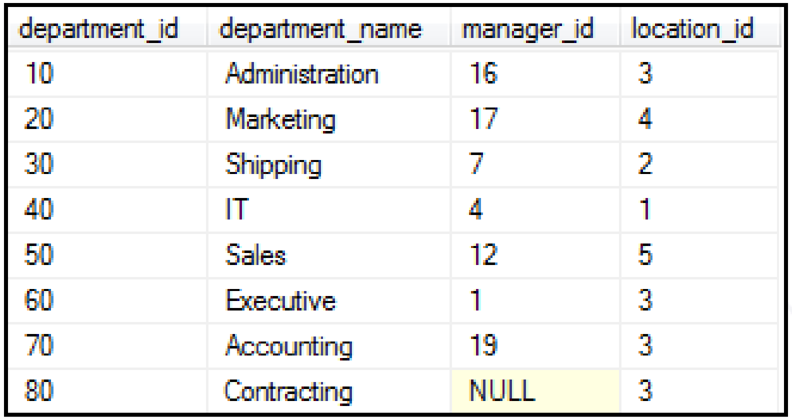
employee



employee_id	first_name	last_name	gender	email	phone_number	hire_date	job_id	salary	commission_pct	manager_id	department_id
1	Steven	King	M	SKING	515 123 4567	1987-06-17	AD_PRES	24000.00	NULL	NULL	60
2	Neena	Kochhar	F	NKOCHHAR	515 123 4568	1989-09-21	AD_VP	17000.00	NULL	1	60
3	Lex	De Haan	M	LDEHAAN	515 123 4569	1993-01-13	AD_VP	17000.00	NULL	1	60
4	Alexander	Hunold	M	AHUNOLD	590 423 4567	1990-01-03	IT_PROG	9000.00	NULL	3	40
5	Bruce	Ernst	M	BERNST	590 423 4568	1991-03-21	IT_PROG	6000.00	NULL	4	40
6	Diana	Lorentz	F	DLORENTZ	590 423 5567	NULL	IT_PROG	4200.00	NULL	4	40
7	Kevin	Mourgos	NULL	KMOURGOS	650 123 5234	1999-05-07	ST_MAN	6000.00	NULL	1	30
8	Trenna	Rajs	F	TRAJS	650 121 8009	1995-10-17	ST_CLERK	3500.00	NULL	7	30
9	Curtis	Davies	M	CDAVIES	650 121 2994	NULL	ST_CLERK	3100.00	NULL	7	30
10	Randall	Matos	NULL	RMATOS	650 121 2874	1998-03-15	ST_CLERK	2600.00	NULL	7	30
11	Peter	Vargas	M	PVARGAS	650 121 2004	1999-05-07	ST_CLERK	2500.00	NULL	7	30
12	Eleni	Zlotkey	F	EZLOTKEY	(011) 44 1344 429018	2000-01-29	SA_MAN	10500.00	0.20	1	50
13	Ellen	Abel	F	EABEL	(011) 44 1644 429267	1996-05-11	SA_REP	11000.00	0.30	12	50
14	Jonathon	Taylor	M	JTAYLOR	(011) 44 1644 429265	1998-03-24	SA_REP	8000.00	0.20	12	50
15	Kimberely	Grant	F	KGRANT	(011) 44 1644 429263	1999-05-07	SA_REP	7000.00	0.15	12	NULL
16	Jennifer	Whalen	F	JWHALEN	515 123 4444	1987-09-17	AD_ASST	4400.00	NULL	2	10
17	Michael	Hartstein	NULL	MHARTSTE	515 123 5555	1996-02-17	MK_MAN	13000.00	NULL	1	20
18	Pat	Fay	F	PFAY	603 123 6646	1997-08-17	MK_REP	3500.00	NULL	17	20
19	Shelley	Higgins	F	SHIGGINS	515 123 8080	1994-06-07	AC_MGR	12000.00	NULL	2	70
20	William	Gietz	M	WGIETZ	515 123 8181	1994-06-07	AC_ACC...	8000.00	NULL	19	70

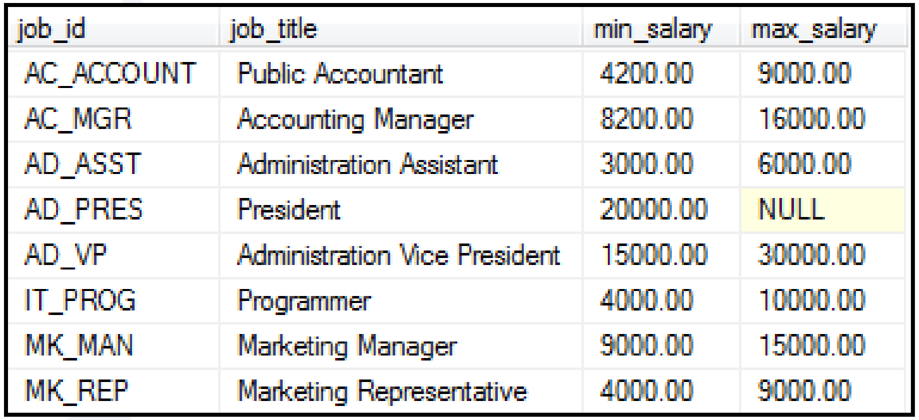
The Company Database

department



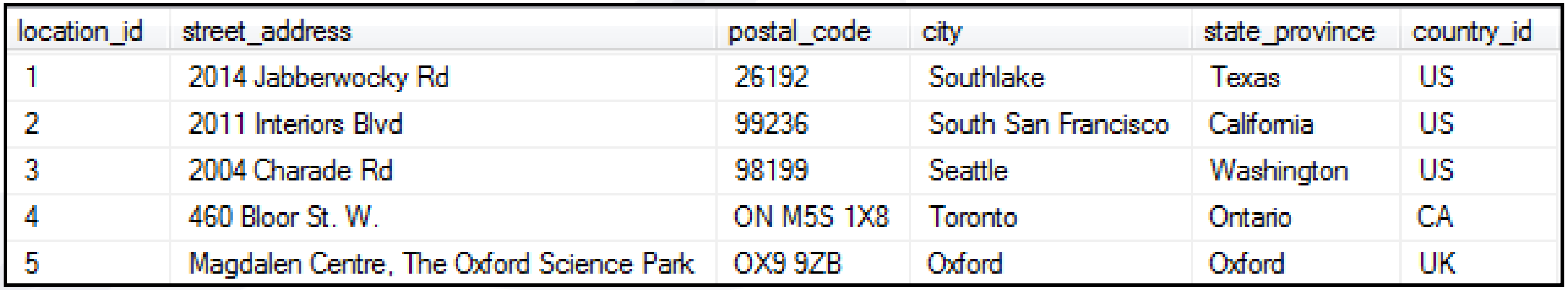
department_id	department_name	manager_id	location_id
10	Administration	16	3
20	Marketing	17	4
30	Shipping	7	2
40	IT	4	1
50	Sales	12	5
60	Executive	1	3
70	Accounting	19	3
80	Contracting	NULL	3

job



job_id	job_title	min_salary	max_salary
AC_ACCOUNT	Public Accountant	4200.00	9000.00
AC_MGR	Accounting Manager	8200.00	16000.00
AD_ASST	Administration Assistant	3000.00	6000.00
AD_PRES	President	20000.00	NULL
AD_VP	Administration Vice President	15000.00	30000.00
IT_PROG	Programmer	4000.00	10000.00
MK_MAN	Marketing Manager	9000.00	15000.00
MK_REP	Marketing Representative	4000.00	9000.00

location

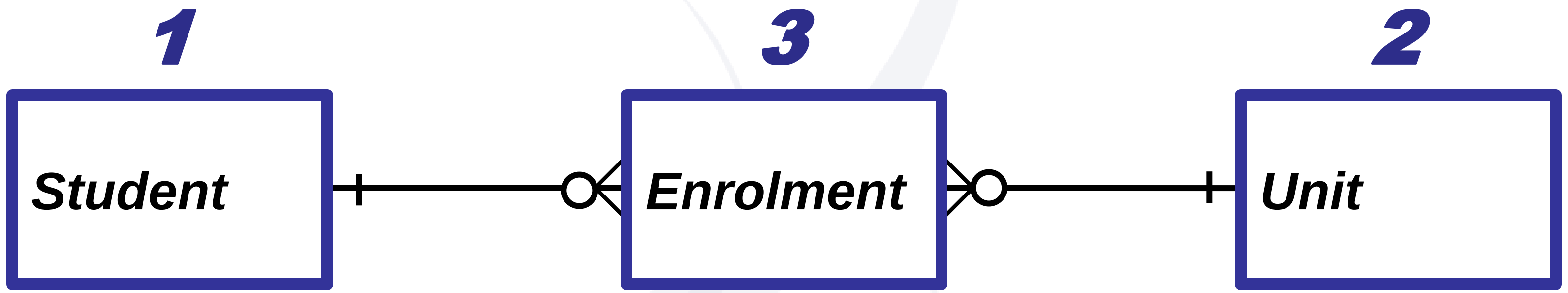


location_id	street_address	postal_code	city	state_province	country_id
1	2014 Jabberwocky Rd	26192	Southlake	Texas	US
2	2011 Interiors Blvd	99236	South San Francisco	California	US
3	2004 Charade Rd	98199	Seattle	Washington	US
4	460 Bloor St. W.	ON M5S 1X8	Toronto	Ontario	CA
5	Magdalen Centre, The Oxford Science Park	OX9 9ZB	Oxford	Oxford	UK

Order of Table Creation

- The basic rule to remember when creating tables is:

  “The one side of a one-to-many relationship must always be made before the many side”



1 Student

3 Enrolment

2 Unit

Student#, Name, Gender, DoB, …

Enrolment#, Student#, UnitCode, Semester, Year, …

UnitCode, UnitTitle, CreditPoints, Description…

- Creating **Unit**, then **Student**, then **Enrolment** would also be appropriate
  - as long as Unit and Student are created before Enrolment

Order of Table Dropping

- Deleting a table in a database is known as **dropping** it

- The rule to remember for this is:

  “All tables with foreign keys be dropped before the table they reference is dropped”

- Essentially…

  “The many side of a one-to-many relationship must always be dropped before the one side”

- In case it isn’t obvious…

  “The drop order is the reverse of the creation order”

Data Dictionaries

- Once a well-structured and normalised database has been designed via normalisation and/or ER modelling, it is almost ready to implement it as an actual database in a DBMS

- The last step in a good design is to create a **data dictionary**

- A data dictionary should contain all the information needed to implement the database in a DBMS. This includes:
  - The names of all entities and their attributes
  - The domain of all attributes (data types, constraints, etc)
  - Details of all primary and foreign keys
  - Written descriptions of entities, attributes, relationships, etc, where needed (e.g. for anything confusing or ambiguous)

Data Dictionaries

- Data dictionaries typically take the form of a number of tables – one table per entity
  - Order the tables in an appropriate table creation order, or remember to specify this information in the data dictionary

“customer” table (stores details about customers)

Column Name	Data Type & Length	Null	Constraints	Other
customer_id	INT	NOT NULL	PK	IDENTITY
name	VARCHAR(50)	NOT NULL		
phone	VARCHAR(20)	NOT NULL		
address	TEXT	NOT NULL		

“order” table (stores details about customers’ orders)

Column Name	Data Type & Length	Null	Constraints	Other
invoice_id	INT	NOT NULL	PK	IDENTITY
order_date	DATETIME	NOT NULL		
customer_id	INT	NOT NULL	FK (customer.customer_id)	

Data Dictionaries

“item” table (stores details about items for sale)

Column Name	Data Type & Length	Null	Constraints	Other
item_id	INT	NOT NULL	PK	IDENTITY
description	VARCHAR(50)	NOT NULL		
unit_price	MONEY	NOT NULL		

“order_item” table (stores details about the items in an order)

Column Name	Data Type & Length	Null	Constraints	Other
invoice_id	INT	NOT NULL	PK, FK (order.invoice_id)	
item_id	INT	NOT NULL	PK, FK (item.item_id)	
qty	SMALLINT	NOT NULL		DEFAULT 1

- A data dictionary should contain *everything* that someone needs to know to implement the database in a DBMS

- Some columns of the data dictionary refer to data types and constraints used in the database/DBMS itself
  - We will cover this in upcoming weeks

Brief Introduction to SQL

- Structured Query Language (SQL) is the language used to send commands to a database in a RDBMS, including…
  - Commands to retrieve data from a database
    - (Standard SQL queries using the “SELECT” command)
  - Commands to insert, update or delete data in a database
    - (Data Manipulation Language - DML)
  - Commands to create, modify and delete database schemas
    - (Data Definition Language – DDL)
  - Commands to manage users access control to a database
    - (Data Control Language – DCL)

- All these languages (DML, DDL, etc) are part of SQL, and have consistent syntax style and structure
  - They are defined only by their purpose

Brief Introduction to SQL

- SQL is a standardised language supported by just about every RDBMS, but many “variations” exist
  - While the common/basic syntax for most commands remains the same, some commands have different syntax
  - They also add features which are often only supported by certain products who have implemented that variation
  - It is unwise to rely heavily on such features, as this limits your ability to transfer your database from one DBMS to another

Common SQL Commands

- Here is a list of some common SQL commands…

Data Retrieval	SELECT

Data Manipulation Language (DML)	INSERT
	UPDATE
	DELETE

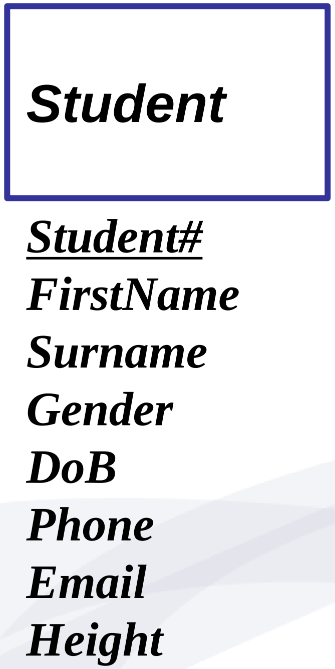
Data Definition Language (DDL)	CREATE DATABASE/TABLE ALTER DATABASE/TABLE DROP DATABASE/TABLE RENAME DATABASE/TABLE TRUNCATE TABLE
Transaction Control	BEGIN/SAVE TRANSACTION COMMIT ROLLBACK
Data Control Language (DCL)	CREATE/ALTER/DROP USER GRANT REVOKE

Creating Tables in SQL

- A table creation statement in SQL consists of several basic elements:
  - The words **CREATE TABLE**
  - The *name* of the table
  - An *opening parenthesis*
  - *Column definitions* (separated by commas)
    - Includes name, data type and other properties of each column
  - *Constraint definitions* (separated by commas)
    - e.g. Key fields, unique fields…
  - A *closing parenthesis*
  - A SQL terminator (**;**)

Creating Tables in SQL

- Remembering that each entity in your physical ERD maps to a table, let’s make a “student” table
  - Some DBMSs do not support “#” in a field name, so we use alphanumeric characters and underscores only



Student

<u>Student#</u>
FirstName
Surname
Gender
DoB
Phone
Email
Height

```
CREATE TABLE student
(
    student_id  INT            NOT NULL    PRIMARY KEY,
    first_name  VARCHAR(20)    NOT NULL,
    surname      VARCHAR(20)   NOT NULL,
    gender      CHAR(1)        NOT NULL,
    dob         DATE           NOT NULL,
    phone       VARCHAR(10)    NULL,
    email       VARCHAR(50)    NULL,
    height      NUMERIC(3,2)   NULL
);
```

Dropping Tables in SQL

- Since we just discussed it, let’s start with something simple: Dropping tables in a database using SQL statements

- The syntax for this statement is simply:
  - **DROP TABLE <table name>;**

- In regards to our last example…

```
DROP TABLE ApplianceOwner;
DROP TABLE Appliance;
DROP TABLE Tenant;
DROP TABLE Unit;
```

- Each statement ends with a **;** (semicolon)

Creating Tables in SQL

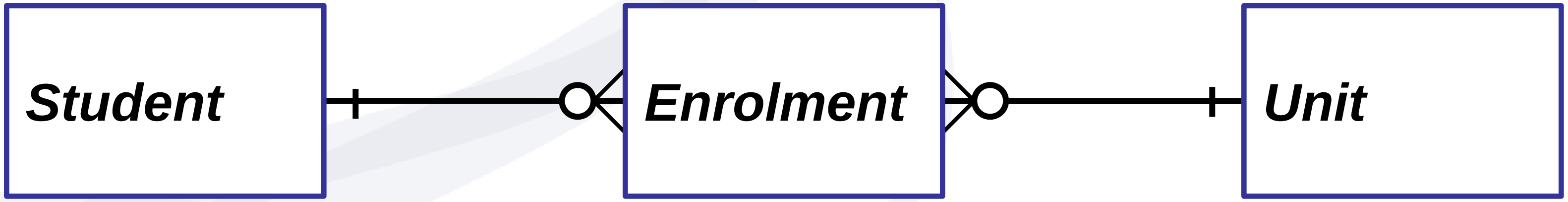
- Here’s another example, this time with a compound key…

```
CREATE TABLE order_item
(
    invoice_id   INT              NOT NULL,
    item_id      CHAR(10)         NOT NULL,
    qty          TINYINT          NOT NULL        DEFAULT 1,
    CONSTRAINT order_item_pk PRIMARY KEY (invoice_id, item_id)
);
```

- Notes:
  - Both the invoice and item ids are part of the primary key (we have omitted foreign keys in this example)
  - Item_id is CHAR(10) rather than an INT… Why?
  - Quantity is TINYINT, which accepts anything from 0 to 255, and it has a default value of 1 – used if no quantity specified

Foreign Keys

- The foreign key, or *referential integrity constraint*, designates a column, or combination of columns, as a foreign key and establishes a relationship to a primary key (or a unique key) in another table (or even the same table).
- Create order must be followed – primary key must already exist in order to create foreign key constraint with it
- Example - create tables for the simple enrolment model:



Student

Enrolment

Unit

StudentID, Name, Gender, DoB

EnrolmentID, StudentID, UnitCode, Semester, Year

UnitCode, UnitTitle, CreditPoints, Description

Foreign Keys

- Student table and Unit table must be created first

```
CREATE TABLE student
(
      student_id      INT                       NOT NULL      PRIMARY KEY,
      name            VARCHAR(50)               NOT NULL,
      gender          CHAR(1)                   NOT NULL,
      dob             DATE                      NOT NULL
);
```

```
CREATE TABLE unit
(
     unit_code        CHAR(7)                   NOT NULL      PRIMARY KEY,
     unit_title       VARCHAR(50)               NOT NULL,
     credit_points    TINYINT                   NOT NULL      DEFAULT 15,
     description      TEXT                      NULL
);
```

Foreign Keys

- And then the enrolments table…

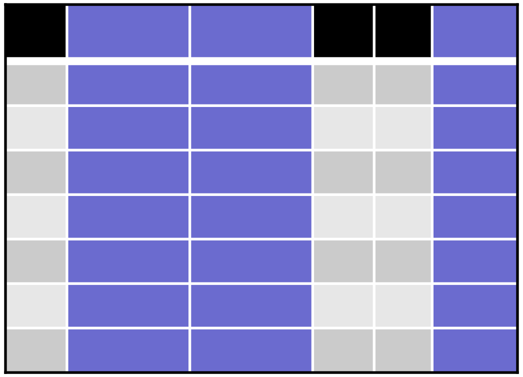
```
CREATE TABLE Enrolment
(
   enrolment_id     INT              NOT NULL    IDENTITY    PRIMARY KEY,
   student_id       INT              NOT NULL,
   unit_code        CHAR(7)          NOT NULL,
   semester         TINYINT          NOT NULL,
   year             SMALLINT         NOT NULL,
   CONSTRAINT stud_fk FOREIGN KEY (student_id) REFERENCES student(student_id),
   CONSTRAINT unit_fk  FOREIGN KEY (unit_code)  REFERENCES unit(unit_code)
);
```

- enrolment_id is primary key, using IDENTITY property to implement an auto-incrementing integer

- Foreign key constraints simply specify the FK field, and the PK field that it refers to (inside the name of its home table)

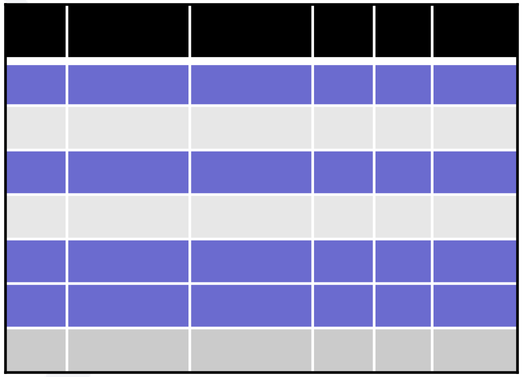
Capabilities of SELECT Statement

Projection

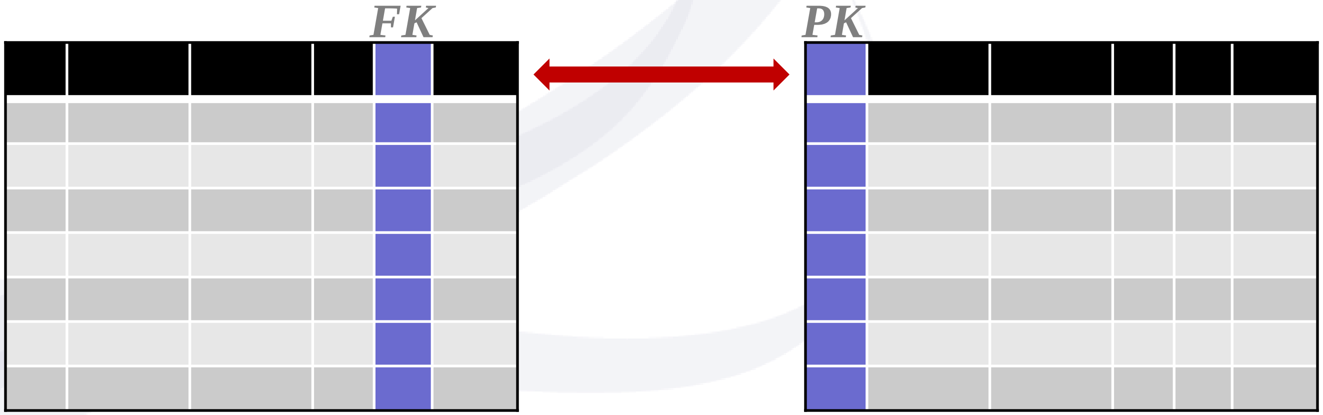


(select certain columns)

Selection



(select certain rows)



Join two tables via keys

Basic SELECT Statement: Syntax

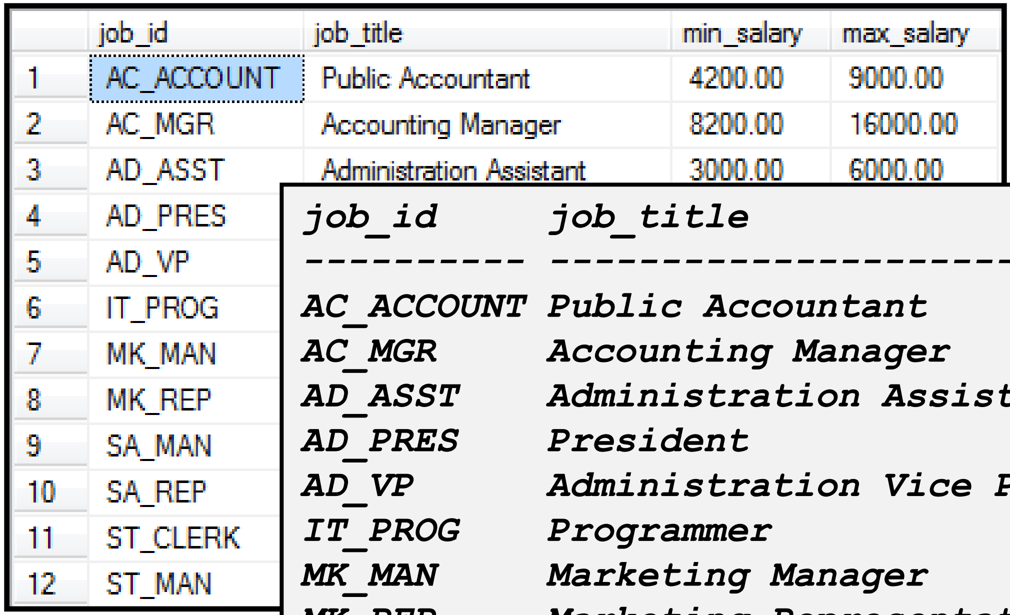
```
SELECT   *|{[DISTINCT] column|expression [alias],...}
FROM     table
[WHERE    Conditions];
```

- SELECT clause identifies *which column(s)*
- FROM clause identifies *which table(s)*
- WHERE clause identifies *which row(s)*
  - Where clause is optional
  - Only those rows whose values make the *conditions* true will be returned

Selecting all columns

```
SELECT *
FROM   job;
```

Grid format result:



	job_id	job_title	min_salary	max_salary
1	AC_ACCOUNT	Public Accountant	4200.00	9000.00
2	AC_MGR	Accounting Manager	8200.00	16000.00
3	AD_ASST	Administration Assistant	3000.00	6000.00
4	AD_PRES			
5	AD_VP			
6	IT_PROG			
7	MK_MAN			
8	MK_REP			
9	SA_MAN			
10	SA_REP			
11	ST_CLERK			
12	ST_MAN			

Text format result:

```
job_id     job_title                       min_salary   max_salary
---------- ------------------------------- ------------ -----------
AC_ACCOUNT Public Accountant               4200.00      9000.00
AC_MGR     Accounting Manager              8200.00      16000.00
AD_ASST    Administration Assistant        3000.00      6000.00
AD_PRES    President                       20000.00     NULL
AD_VP      Administration Vice President   15000.00     30000.00
IT_PROG    Programmer                      4000.00      10000.00
MK_MAN     Marketing Manager               9000.00      15000.00
MK_REP     Marketing Representative        4000.00      9000.00
SA_MAN     Sales Manager                   10000.00     20000.00
SA_REP     Sales Representative            6000.00      12000.00
ST_CLERK   Stock Clerk                     1000.00      5000.00
ST_MAN     Stock Manager                   5500.00      8500.00

(12 row(s) affected)
```

Selecting Specific Columns

```
SELECT job_id, max_salary
FROM   job;
```

```
job_id     max_salary
---------- ----------
AC_ACCOUNT 9000.00
AC_MGR     16000.00
AD_ASST    6000.00
AD_PRES    NULL
AD_VP      30000.00
IT_PROG    10000.00
MK_MAN     15000.00
MK_REP     9000.00
SA_MAN     20000.00
SA_REP     12000.00
ST_CLERK   5000.00
ST_MAN     8500.00

(12 row(s) affected)
```

Guideline for writing SQL statements

- Goals: correct syntax, readability and easy to edit
  - SQL statements are not case sensitive
  - SQL statements can be on one or more lines
  - Keywords cannot be abbreviated or split across lines
  - Keywords are typically entered in UPPERCASE
  - New clauses are usually placed on separate lines
  - Indents are used to enhance readability

Arithmetic Expressions

- Can create expressions with number and date data by using arithmetic operators.
  - Add (+), subtract (-), multiply (*) and divide (/)
  - Other operators exist, but these are the most common
- Operator Precedence:
  - Multiplication and division take priority over addition and subtraction
  - Operators of the same priority are evaluated from left to right
  - Parentheses are used to prioritise evaluation and to clarify statements

Arithmetic Expressions

```
SELECT job_id, min_salary, max_salary, max_salary*1.05
FROM   job;
```

```
job_id     min_salary  max_salary
---------- ----------- ----------- ----------
AC_ACCOUNT 4200.00     9000.00     9450.000000
AC_MGR     8200.00     16000.00    16800.000000
AD_ASST    3000.00     6000.00     6300.000000
AD_PRES    20000.00    NULL        NULL
AD_VP      15000.00    30000.00    31500.000000
IT_PROG    4000.00     10000.00    10500.000000
MK_MAN     9000.00     15000.00    15750.000000
MK_REP     4000.00     9000.00     9450.000000
SA_MAN     10000.00    20000.00    21000.000000
SA_REP     6000.00     12000.00    12600.000000
ST_CLERK   1000.00     5000.00     5250.000000
ST_MAN     5500.00     8500.00     8925.000000

(12 row(s) affected)
```

No column name for “max_salary*1.05”

Arithmetic Expressions

```
SELECT job_id, min_salary, max_salary, max_salary*1.05
FROM   job;
```

```
job_id     min_salary  max_salary
---------- ----------- ----------- ------------
AC_ACCOUNT 4200.00     9000.00     9450.000000
AC_MGR     8200.00     16000.00    16800.000000
AD_ASST    3000.00     6000.00     6300.000000
AD_PRES    20000.00    NULL        NULL
AD_VP      15000.00    30000.00    31500.000000
IT_PROG    4000.00     10000.00    10500.000000
MK_MAN     9000.00     15000.00    15750.000000
MK_REP     4000.00     9000.00     9450.000000
SA_MAN     10000.00    20000.00    21000.000000
SA_REP     6000.00     12000.00    12600.000000
ST_CLERK   1000.00     5000.00     5250.000000
ST_MAN     5500.00     8500.00     8925.000000

(12 row(s) affected)
```

- A **NULL** is a value that is *unavailable*, *unassigned*, *unknown*, or *inapplicable*
- A null is ***not** the same as* zero or a blank space
- Arithmetic expressions containing a null value evaluate to null

Column Aliases

- A column alias:
  - Renames a column heading for the results of that query
  - Is useful with calculations and other situations where column names may be missing, unhelpful or ambiguous

- Usual form is **`AS 'aliasname'`** after the column name
  - The AS is optional, but recommended for clarity

  - If the alias contains spaces or special characters, you must enclose it in single quotation marks
    - If the alias is a single word with no special characters, the quote marks can be omitted

Column Aliases

```
SELECT job_title AS Job, max_salary AS 'Pre-raise Maximum',
       max_salary*1.05 AS 'Post-raise Maximum'
FROM   job;
```

```
Job                            Pre-raise Maximum  Post-raise Maximum
------------------------------ ------------------ ------------------
Public Accountant              9000.00            9450.000000
Accounting Manager             16000.00           16800.000000
Administration Assistant       6000.00            6300.000000
President                      NULL               NULL
Administration Vice President  30000.00           31500.000000
Programmer                     10000.00           10500.000000
Marketing Manager              15000.00           15750.000000
Marketing Representative       9000.00            9450.000000
Sales Manager                  20000.00           21000.000000
Sales Representative           12000.00           12600.000000
Stock Clerk                    5000.00            5250.000000
Stock Manager                  8500.00            8925.000000

(12 row(s) affected)
```

Concatenation

- A function named CONCAT() allows you to join multiple text-based columns into a single column of results

```
SELECT CONCAT(job_id, job_title) AS 'Job ID & Title'
FROM   job;
```

```
Job ID & Title
-------------------------------------
AC_MGRAccounting Manager
AD_ASSTAdministration Assistant
AD_VPAdministration Vice President
MK_MANMarketing Manager
MK_REPMarketing Representative
AD_PRESPresident
IT_PROGProgrammer
AC_ACCOUNTPublic Accountant
SA_MANSales Manager
SA_REPSales Representative
ST_CLERKStock Clerk
ST_MANStock Manager

(12 row(s) affected)
```

Concatenation

- You can add other text to this *in single quotes* **as needed**

```
SELECT CONCAT(job_id, ' (', job_title, ')') AS 'Job ID & Title'
FROM   job;
```

```
Job ID & Title
-------------------------------------
AC_MGR (Accounting Manager)
AD_ASST (Administration Assistant)
AD_VP (Administration Vice President)
MK_MAN (Marketing Manager)
MK_REP (Marketing Representative)
AD_PRES (President)
IT_PROG (Programmer)
AC_ACCOUNT (Public Accountant)
SA_MAN (Sales Manager)
SA_REP (Sales Representative)
ST_CLERK (Stock Clerk)
ST_MAN (Stock Manager)

(12 row(s) affected)
```

Concatenation

- A common use of concatenation is to produce full names…

```
SELECT CONCAT(first_name, ' ', last_name) AS 'full_name'
FROM   employee;
```

```
Full_name
-------------------------------------
Steven King
Neena Kochhar
Lex De Haan
Alexander Hunold
Bruce Ernst
...

 (20 row(s) affected)
```

- *In earlier versions of SQL Server, "+" was used to concatenate. This is still supported (only in SQL Server)*

```
SELECT first_name + ' ' + last_name AS 'full_name'
FROM   employee;
```

Duplicate Rows & the DISTINCT Keyword

- *By default, a query will display all rows, including rows which contain the same values (i.e. duplicates)*

```
SELECT job_id
FROM   employee;
```

```
job_id
-----------
AD_PRES
AD_VP
AD_VP
IT_PROG
IT_PROG
IT_PROG
ST_MAN
ST_CLERK
ST_CLERK
...
(20 row(s) affected)
```

Duplicate Rows & the DISTINCT Keyword

- *You can eliminate duplicate rows by using the DISTINCT keyword in the SELECT clause.*

```
SELECT DISTINCT job_id
FROM   employee;
```

```
job_id
----------
AC_ACCOUNT
AC_MGR
AD_ASST
AD_PRES
AD_VP
IT_PROG
MK_MAN
MK_REP
SA_MAN
SA_REP
ST_CLERK
ST_MAN

(12 row(s) affected)
```

Summary of SELECT Basics

- That covers the basics of the SELECT statement
  - Selecting all columns with *
  - Specifying which columns to select by column name
  - Arithmetic operators
  - Column aliases
  - Concatenation
  - Using the DISTINCT keyword to eliminate duplicates

- These can all be combined in order to select something very specific from a table

- In coming weeks, we will learn to specify criteria with the WHERE clause, and connect columns from different tables using joins

Microsoft SQL Server Overview

- Microsoft SQL Server includes multiple components and services that makes it a comprehensive enterprise platform

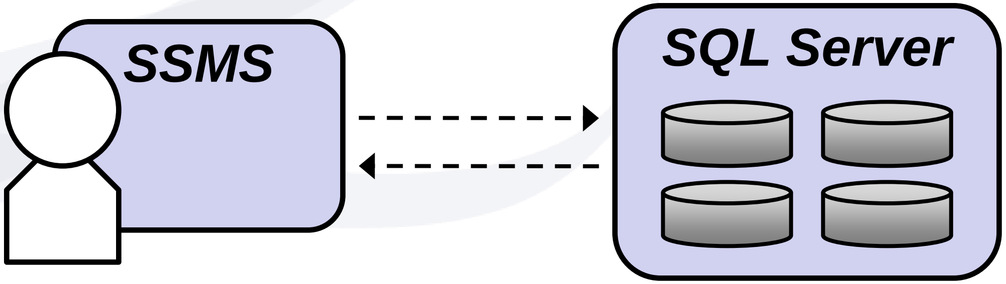
- Key Components:
  - Database Engine
  - Analysis Services
  - Integration Services
  - Replication Services
  - Reporting Services
  - ***Service Broker***
  - ***Native HTTP Support***
  - ***.NET Common Language Runtime (SQL CLR)***
  - ***Notification Services***
  - ***Full-Text Search, and more!***

- Newer versions of SQL Server exist, but the implementation of SQL in the newer versions is almost identical
  - This unit focuses on the SQL, not the other server features

SQL Server Management Studio (SSMS)

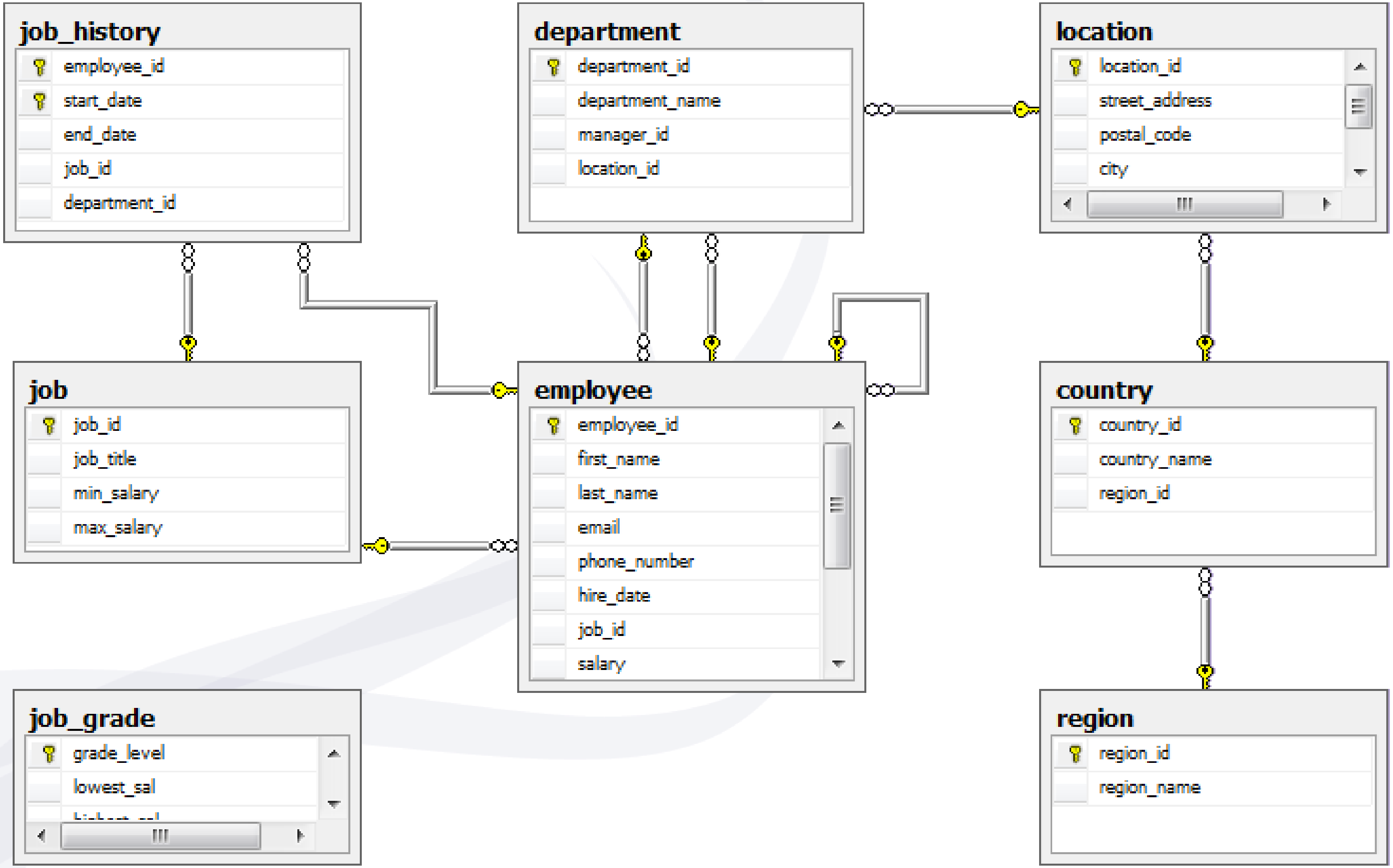
- SSMS is an integrated database management and development environment for SQL Server

- Developers can use it to
  - Create databases, tables and other objects
  - Change table structures and constraints
  - Create, execute and save scripts/queries
  - Manage databases, services and other components
  - Create database diagrams



Company ER Diagram in SSMS

- SSMS can generate database diagrams such as this...



job_history
employee_id
start_date
end_date
job_id
department_id

department
department_id
department_name
manager_id
location_id

location
location_id
street_address
postal_code
city

job
job_id
job_title
min_salary
max_salary

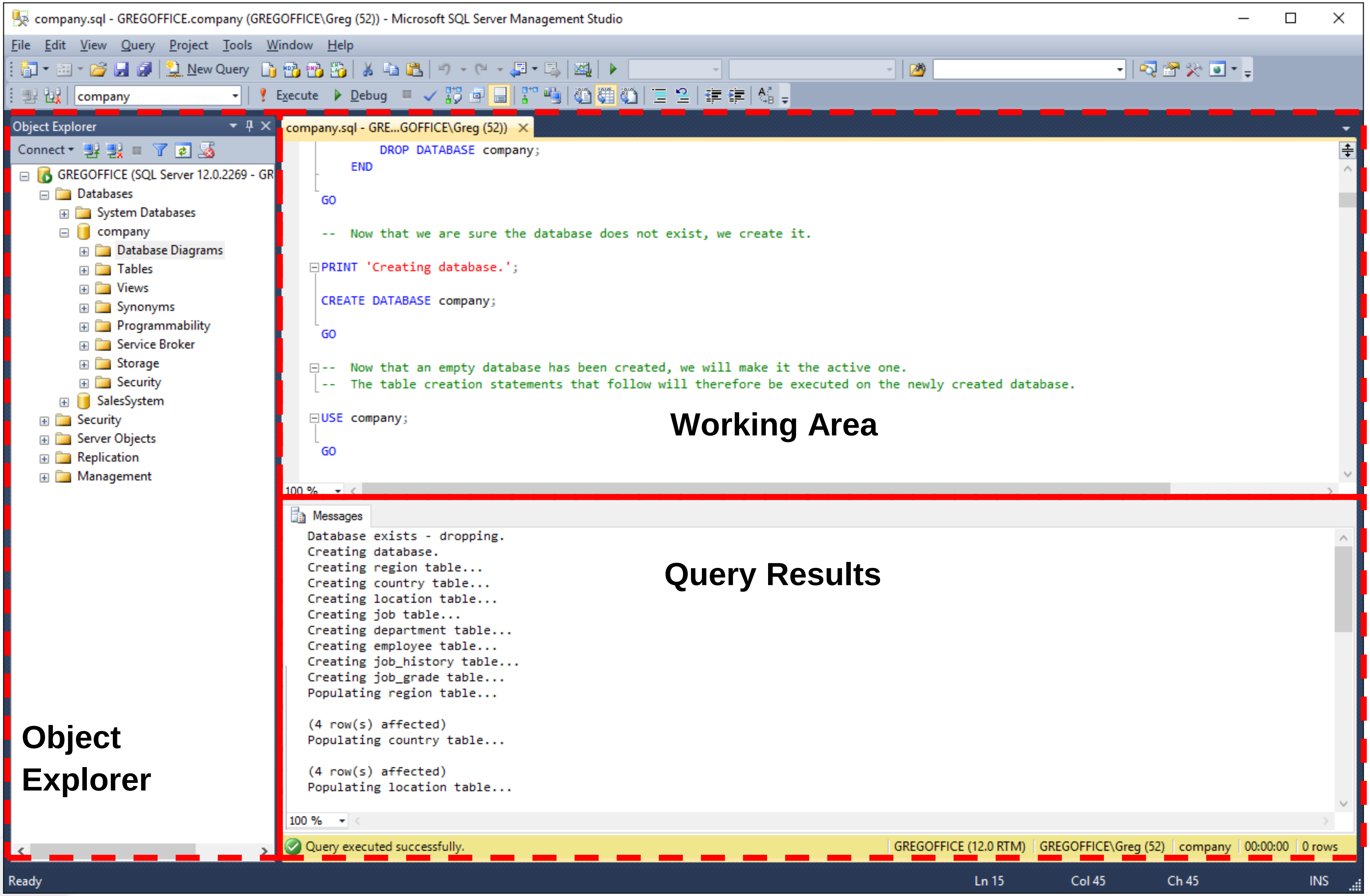
employee
employee_id
first_name
last_name
email
phone_number
hire_date
job_id
salary

country
country_id
country_name
region_id

job_grade
grade_level
lowest_sal
highest_sal

region
region_id
region_name

SQL Server Management Studio (SSMS)



```
        DROP DATABASE company;
    END

GO

-- Now that we are sure the database does not exist, we create it.

PRINT 'Creating database.';

CREATE DATABASE company;

GO

-- Now that an empty database has been created, we will make it the active one.
-- The table creation statements that follow will therefore be executed on the newly created database.

USE company;

GO
```

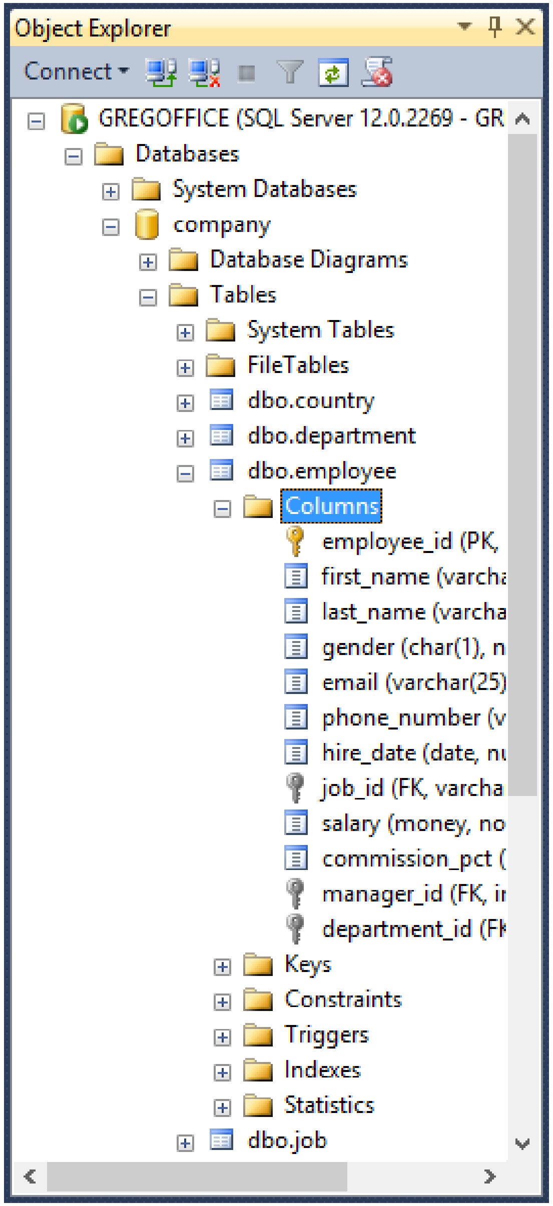
```
Database exists - dropping.
Creating database.
Creating region table...
Creating country table...
Creating location table...
Creating job table...
Creating department table...
Creating employee table...
Creating job_history table...
Creating job_grade table...
Populating region table...

(4 row(s) affected)
Populating country table...

(4 row(s) affected)
Populating location table...
```

SSMS – Object Explorer

- The left menu is the **Object Explorer**
- **Browse**, access and manage all objects (tables, databases, etc) on the server
- **Refresh button** may be needed before new items show up
- **Right click** on objects for menu of useful/common commands
  - Right click on a table (or view) to select or edit its contents
  - Right click on column to edit data type, length, null, etc



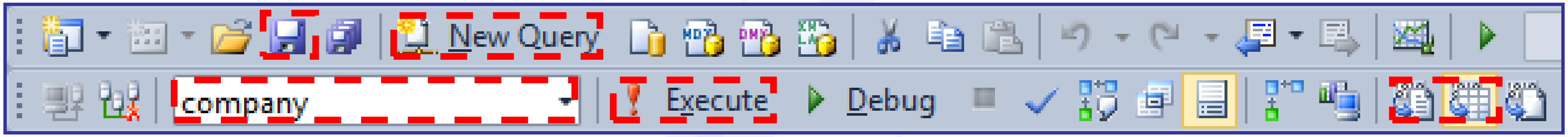
- GREGOFFICE (SQL Server 12.0.2269 - GR
  - Databases
    - System Databases
    - company
      - Database Diagrams
      - Tables
        - System Tables
        - FileTables
        - dbo.country
        - dbo.department
        - dbo.employee
          - Columns
            - employee_id (PK,
            - first_name (varcha
            - last_name (varcha
            - gender (char(1), n
            - email (varchar(25)
            - phone_number (v
            - hire_date (date, nu
            - job_id (FK, varcha
            - salary (money, no
            - commission_pct (
            - manager_id (FK, in
            - department_id (FK
          - Keys
          - Constraints
          - Triggers
          - Indexes
          - Statistics
        - dbo.job

SSMS – Useful Toolbar Items

- Many of the toolbar buttons are very useful, but here are a few you will probably use the most frequently
  - Some toolbars will only appear while working on a query
  - Hover over other buttons to see tooltip about their purpose

Save active tab *in the work area*

Open new query *tab* in work area



Select active database

Run (selected part of or whole) query

Show results as table or text

- **Note:** If you have some text selected in your query, *only the selected text* will be executed when you press the execute button

Starting SQL Server Management Studio

- In ECU labs:
  - Find it by searching for it in the start menu (type “SQL”) and connect with the default settings (just press Connect button)
  - *Follow the instructions in Tutorial 7 to get started this week*

- At home:
  - *Follow the installation guide in Canvas*
  - Download and install SQL Server 2014 Express Edition
    - This is a free version of SQL Server 2014
    - Make sure you install Management Studio
    - Once installed, launch Management Studio and connect with the default settings

- Once set up, you’re ready to play with some databases!

Summary

Summary

- Introduction to SQL
- Basics of the SELECT statement
- Introduction to SQL Server and SQL Server Management Studio (SSMS)
- From this week onwards the workshop sessions are labs, in which we work in SSMS

Questions?