Database Development: more on keys, scripts constraints and code.

Lecturer: Dr. Richard Holden

Topics this week

- **Recap and demo of some important points about keys**
  - Referential integrity
- **Another simpler (than coursework) script system**
  - Order calling and key constraints
- **Recap and then idea of embedded code**

Keys again

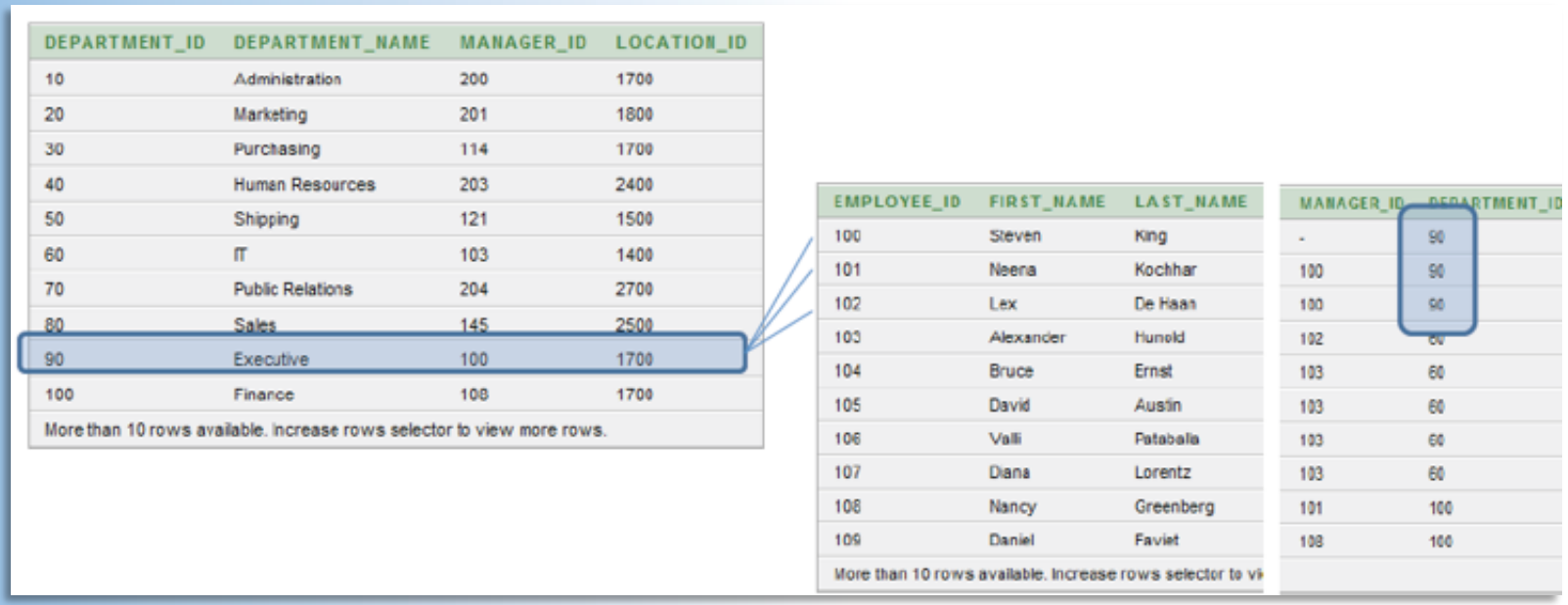
1. Recap
2. Some script examples
3. Implications of keys, generally
4. And implications of keys for script-call order

Keys again

- Identification purposes
- An attribute or combination of attribute values (composite key) which will always produce unique values which are used to distinguish between entities
- A key must always have a value for an entity instance
  - Set when an entity instance is created
- **Candidate** keys are single attributes or combination of attributes which can uniquely identify an entity instance
- **Primary** key is the commonly used way of uniquely identifying an entity instance i.e. selected Candidate key
- **Foreign** keys implement relationships between entity instances

Relationships: one to many

- A department has many employees
  - i.e. a one-to-many (1:N) relationship between Departments and Employees
- To implement: Primary Key of Departments is included as a Foreign Key within Employees



DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	121	1500
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700

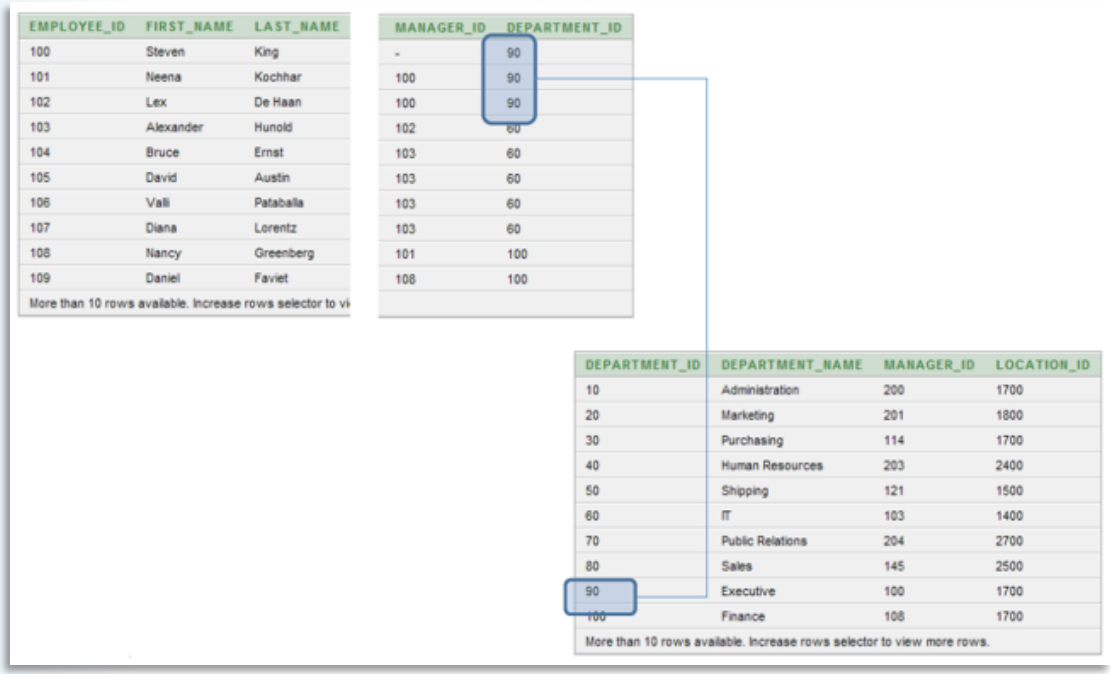
More than 10 rows available. Increase rows selector to view more rows.

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	MANAGER_ID	DEPARTMENT_ID
100	Steven	King	-	90
101	Neena	Kochhar	100	90
102	Lex	De Haan	100	90
103	Alexander	Hunold	102	60
104	Bruce	Ernst	103	60
105	David	Austin	103	60
106	Valli	Pataballa	103	60
107	Diana	Lorentz	103	60
108	Nancy	Greenberg	101	100
109	Daniel	Faviet	108	100

More than 10 rows available. Increase rows selector to vi

Relationships: one to many

- An employee has only one department
- Which means any given tuple in the employee table has one value for the department



EMPLOYEE_ID	FIRST_NAME	LAST_NAME
100	Steven	King
101	Neena	Kochhar
102	Lex	De Haan
103	Alexander	Hunold
104	Bruce	Ernst
105	David	Austin
106	Valli	Pataballa
107	Diana	Lorentz
108	Nancy	Greenberg
109	Daniel	Faviet

More than 10 rows available. Increase rows selector to vi

MANAGER_ID	DEPARTMENT_ID
-	90
100	90
100	90
102	60
103	60
103	60
103	60
103	60
101	100
108	100

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	121	1500
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700

More than 10 rows available. Increase rows selector to view more rows.

Keys force dependency between scripts

- In relation to the scripts:
  - The examples
  - And the ones you have been developing in the labs
- We talked about functional dependency, previously
  - B is said to be **functionally dependent** on A if each value of A is associated with exactly one value of B at any instant in time etc
- Think of dependency between you data insertions too, as discussed in the tutorials

Keys force dependency between scripts

- Think of dependency between you data insertions too, as discussed in the tutorials
  - i.e., dependency between you load-*.sql files

```
.system cls
.system echo "inside top-level.sql"
--
PRAGMA foreign_keys = ON;
.read create-department.sql
.read create-employee.sql

--/*
.system echo "Expects no failure in this order"
.read load-department.sql
.read load-employee.sql
.read sql-commands.sql
--*/
```

Keys force dependency between scripts

- Think of dependency between your data insertions too, as discussed in the tutorials
  - i.e., dependency between your load-*.sql files
- Nothing intrinsically wrong with the following:

```
.system echo "Expects no failure in this order"
.read load-department.sql
.read load-employee.sql
```

Keys force dependency between scripts

- So, why might we expect it to fail?
- It depends on the relationships (established by any keys in the relevant tables)

```
.system echo "Expects failure in this order"
.read load-employee.sql
.read load-department.sql
```

Keys force dependency between scripts

- Why does this fail?

```
.system echo "Expects failure in this order"
.read load-employee.sql
.read load-department.sql
```

- This is why →
- This is why →

```
CREATE TABLE IF NOT EXISTS Employee (
    employeeId INTEGER PRIMARY KEY,
    name TEXT NOT NULL,
    sname TEXT NOT NULL,
    departmentId INTEGER,-- PASS
    -- departmentId INTEGER DEFAULT 0,-- FAILS
    FOREIGN KEY(departmentId) REFERENCES Department(departmentId)
        ON DELETE CASCADE
);
```

```
INSERT INTO Employee(name, sname, departmentId)
    VALUES
        ("Neena", "Kochhar", 9),
        ("Lex", "De Hass", 9),
        ("Alexandra", "Hunold", 6),
        ("Bruce", "Ernst", 6),
        ("David", "Austin", 6),
        ("Valli", "Patabala", 6),
```

Keys force dependency between scripts

- So, why might we expect it to fail?
- It depends on the relationships (established by any keys in the relevant tables)

```
.system echo "Expects no failure in this order"
.read load-department.sql
.read load-employee.sql
```

Some SQL (sql-commands.sql)

- SELECT * FROM etc
- Lets look at some newer stuff in bottom half of script….

```
.system echo "*********************************"
.system echo "in sql-commands.sql"
.system echo "Some  SQL Commands"
.system echo "------------------------------"


.system echo "1 All data from Department table:"
.system echo "------------------------------"
SELECT * FROM Department;

.system echo "2 All data from Employee table:"
.system echo "------------------------------"
SELECT * FROM Employee;

.system echo "3 All data from Manager table:"
.system echo "------------------------------"
SELECT * FROM Manager;

.system echo "4 Which employees have a manager?"
.system echo "------------------------------"
SELECT name, sname FROM Employee
    WHERE departmentId IN (SELECT departmentId FROM Manager);

.system echo "5 Which employees have a manager and who is it?"
.system echo "------------------------------"
SELECT Employee.name, Employee.sname, Manager.name, Manager.sname
FROM Employee
INNER JOIN Manager
ON Employee.departmentId = Manager.departmentId;

.system echo "6 Joining employees with departments"
.system echo "------------------------------"
SELECT Employee.name, Employee.sname, Department.name
FROM Employee
INNER JOIN Department
ON Employee.departmentId = Department.departmentId;


.system echo "7 Storing a view of the join query then displaying it..."
.system echo "------------------------------"
DROP VIEW IF EXISTS EmployeeDepartments;
CREATE VIEW EmployeeDepartments AS
SELECT Employee.name, Employee.sname, Department.name
FROM Employee
INNER JOIN Department
ON Employee.departmentId = Department.departmentId;

SELECT * FROM EmployeeDepartments;
```

Some SQL (sql-commands.sql)

```
.system echo "4 Which employees have a manager?"
.system echo "-----------------------------"
SELECT name, sname FROM Employee
    WHERE departmentId IN (SELECT departmentId FROM Manager);

.system echo "5 Which employees have a manager and who is it?"
.system echo "-----------------------------"
SELECT Employee.name, Employee.sname, Manager.name, Manager.sname
FROM Employee
INNER JOIN Manager
ON Employee.departmentId = Manager.departmentId;
```

Some SQL (sql-commands.sql)

```
.system echo "6 Joining employees with departments"
.system echo "------------------------------"
SELECT Employee.name, Employee.sname, Department.name
FROM Employee
INNER JOIN Department
ON Employee.departmentId = Department.departmentId;


.system echo "7 Storing a view of the join query then displaying it..."
.system echo "------------------------------"
DROP VIEW IF EXISTS EmployeeDepartments;
CREATE VIEW EmployeeDepartments AS
SELECT Employee.name, Employee.sname, Department.name
FROM Employee
INNER JOIN Department
ON Employee.departmentId = Department.departmentId;

SELECT * FROM EmployeeDepartments;
```

Bit of a course recap + embedded code/sql

GCU
Glasgow Caledonian University

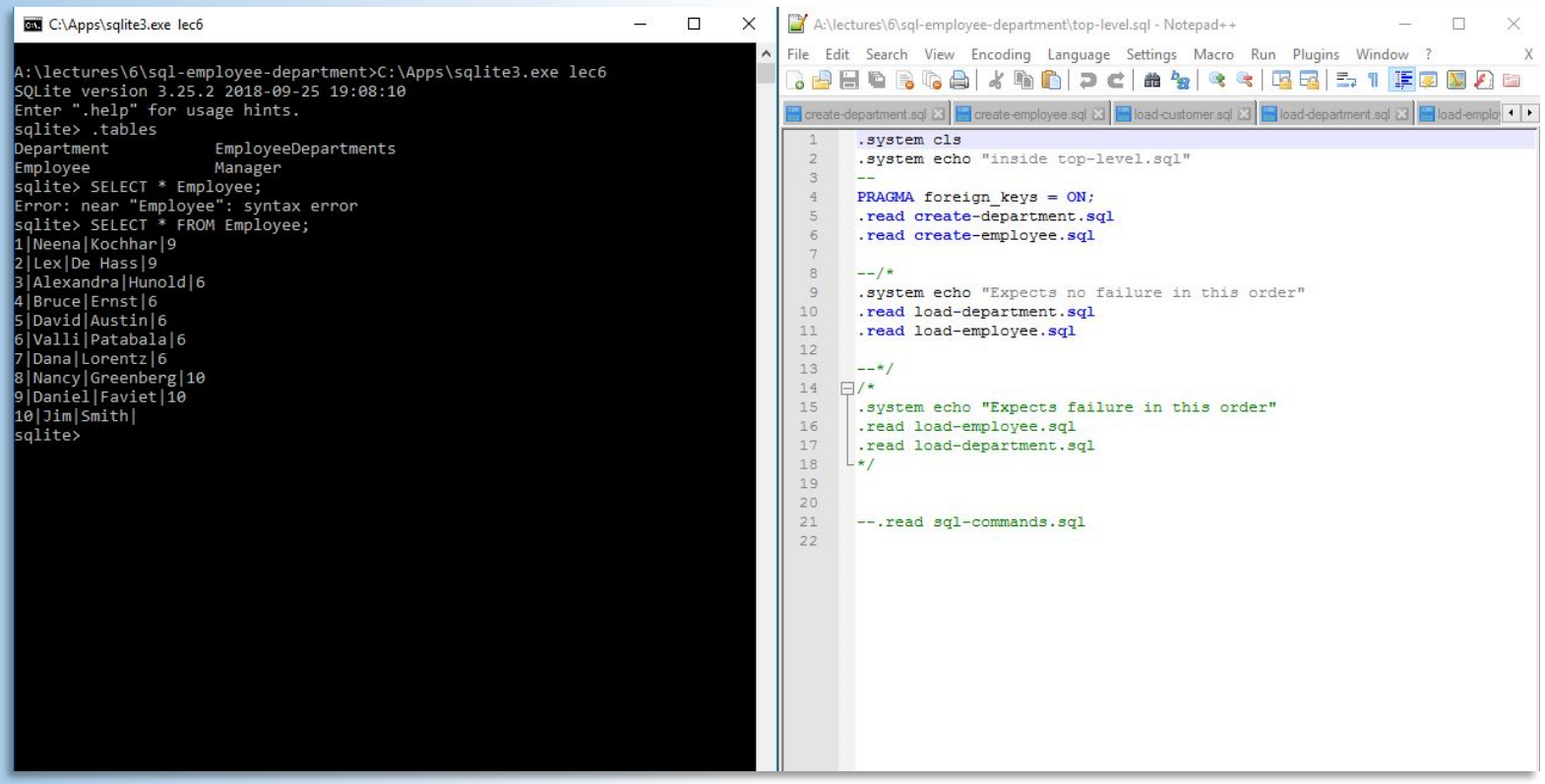
Various content, including:

GCU Glasgow Caledonian University

- Importance of context: data – structured to unstructured
  - Different structures imply different programming
  - Relational modelling is at the structured side of data modelling
- RDBM - Designing for data: schema, relations, keys, ERD, top-down, bottom-up (0-to-3 NF)
- Implementation: environment, SQL.
  - Earlier parts focus on environment, then impl of schema, script environment.
  - Non-gui, script-based environment (forces learning the command strings)

Environment

Environment



Iteration

Frequent building/running
Script development

Environment

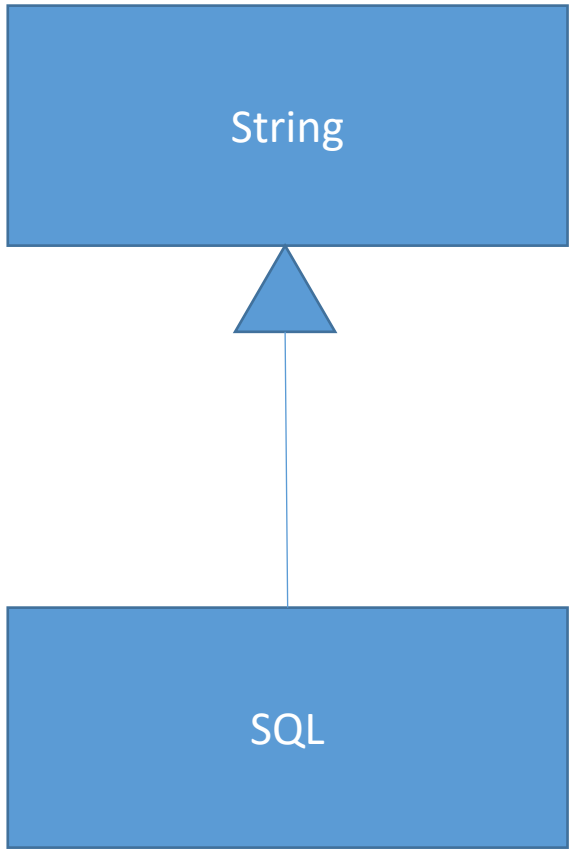
- Why non-GUI?

  Any sqlite command:
  - in cmd
  - in .sql script

  …If you think about it

  ..**is a** String

- This brings us to the ***general*** idea of ‘embedding code’.



Embedded code

- By which we mean code in code
  - Phython in java (e.g. https://www.jython.org/)
  - php in html
  - Javascript in html
- Next week some Java
  - So, sql in Java

- String sql = “YOUR SQL CODE”;