UN-EDITED STUDENT ANSWER

1. Attributes and Assumptions

1.1 Introduction.

Below there is a list of all entities specified in the coursework and the attributes they would initially hold.

It would be desirable to store references provided by the employee during the selection process. Database should store reference to the paper record storing references provided by the candidate employee.

Text in RED indicates primary keys in this section.

1.2 Employee table.

Attribute Name	Description	Type
emp_id	Unique number identifying each employee.	varchar2 (4)
fname	First name of the employee.	varchar2 (15)
sname	Surname of the employee.	varchar2 (15)
DoB	Date of birth of the employee.	date
sex	Gender of the employee; it can take only values of 'M' (= male) or 'F' (=female).	char (1)
telephone	Telephone number of the employee.	number (11)
hname	Information concerned with the postal address (house number or name).	varchar2 (20)
street	Information concerned with the postal address.	varchar2 (20)
town	Information concerned with the postal address.	varchar2 (20)
postcode	Information concerned with the postal address.	varchar2 (8)
b_sortcode	Sort code of the bank account owned by the employee (often identifies branch of the bank)	number (6)
b_accno	Number of the account owned by the employee.	number (8)
b_name	Name of the bank of the employee.	varchar2 (15)
p_method	Preferred payment method (ie. cheque, bank transfer, cash) {'Q' | 'T' | 'C'}	char (1)
r_id	Unique id number identifying reference provided by the employee.	varchar2 (4)
r_name	Name of person referring this employee during the recruitment process.	varchar2 (30)
r_location	Location of that reference in the paper record.	varchar2 (4)

Primary key: emp_id is an artificially created primary key; it is better than any other possible candidate key.

1.3 PromotionHistory table.

Attribute Name	Description	Type
emp_id	Unique number identifying each employee (foreign key).	varchar2 (4)
prom_date	Promotion date.	date
position	Position to which employee is promoted.	varchar2 (20)
department	Department in which this position is available.	varchar2 (20)
salary	Salary allocated to a position in a department.	number (6)
mod	Modification to the salary.	number (6)
description	Description of the position to which employee is promoted.	varchar2 (30)
narrative	Comments regarding this promotion. Not required.	varchar2 (30)

As it is possible that an employee is promoted twice to the same position (employee is promoted, then demoted and then promoted to the same position), this information cannot be used as part of the primary key.

Combination of emp_id and prom_date provides enough information to uniquely identify each record in this table.

1.4 AbsenceHistory table.

Attribute Name	Description	Type
emp_id	Unique number identifying each employee (foreign key).	varchar2 (4)
abs_start	Date when absence started.	date
abs_end	Date when absence ended.	date
code	Code of the absence reason.	varchar2 (2)
reason	Reason of the absence (translation of the code)	varchar2 (30)
paid	Flag indicating if employee s going to be paid for that absence period ('Y' or 'N')	char (1)

Combination of emp_id and abs_start provides enough information to identify each record in this table.

abs_end can be equal to null – in normal circumstances entry in this table is created with abs_end equal to null. Subsequently this entry will be updated when the employee comes back.

1.5 TrainingHistory table.

Attribute Name	Description	Type
emp_id	Unique number identifying each employee (foreign key).	varchar2 (4)
c_start	Date when course starts.	date
c_end	Date when course ends.	date
c_name	Unique name identifying training course.	varchar2 (30)
description	Description of the course.	varchar2 (50)
organiser	Name of the company/individual organising the course.	varchar2 (20)
org_hnumber	First line of the address of the organiser of the course.	varchar2 (20)
org_street	Second line of the address of the organiser of the course.	varchar2 (20)
org_town	Town in which organiser is located.	varchar2 (20)
org_postcode	Postcode of the organiser.	varchar2 (8)
location	Location where course is performed.	varchar2 (50)
result	Result of the training.	varchar2 (30)

Primary key in this table is a combination of three attributes. There are no other shorter candidate keys in this table. emp_id, c_start and c_name uniquely identify each entry in this table. Course name is part of the primary key to allow employees to start multiple courses at the same date.

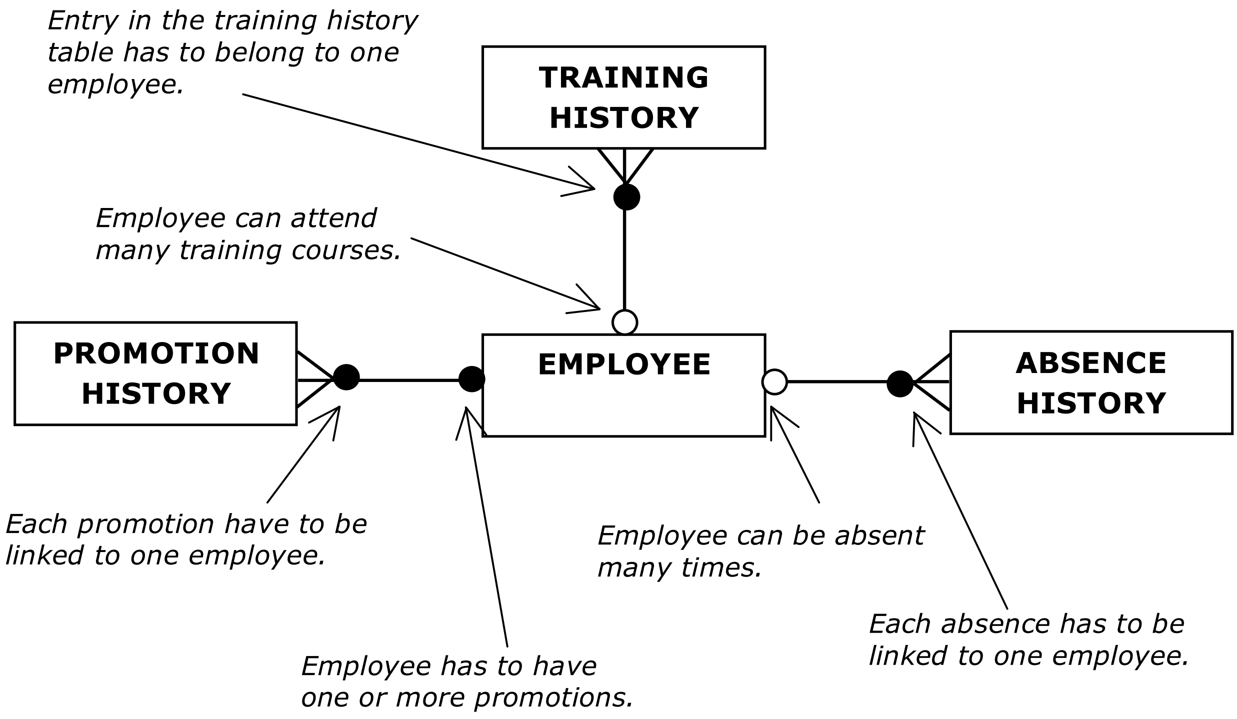
1.6 Assumptions.

Following assumptions need to be satisfied:

- When new employee is added to the database, an entry in the PromotionHistory is created to indicate in what position that person is going to work.
- Employee can be promoted only once a day. It doesn't make sense to promote employee more than once a day. From the database point of view promotion can be from any position to any position (added flexibility).
- Employee cannot be absent for less than one second – Oracle DATE type can store time with accuracy up to one second. From the practical point of view this constraint cannot be broken during normal usage of the database; probably it will never be exercised.
- Employee can start attending more than one training course at the date, given that they are different training courses (ie. cannot start to attend two the same training courses on the same dates). Clearly there will be employees who will attend multiple courses at the same time, possibly starting at the same date. It doesn't make sense for the employee to start attending 2 same training courses.
- Initially it should be assumed that c_names are unique. By assuming that we can identify courses by their names. In case that this assumption is not valid anymore c_id should be introduced.
- Name of the organiser is uniquely identifying company organising a training course. By assuming that we can identify companies organising courses by their names.
- Employee can provide zero to many references during the employment process. Normally candidate employee has to provide some references during the recruitment; this information should be recorded in the database.
- Paid field depends on the code (of the absence reason). It is reasonable to assume that company has a policy regarding paying it's employees during their absences.
- Base salary depends on the position and department. It makes sense to assume that people doing similar jobs will have the same pay.
- Base salary can be individually modified; this information will be stored in the promotionhistory table in form of a signed number. Default value will be 0.

- Employee cannot be older than 130 years. Creating this constraint should limit number of possible errors.
- It is possible to create record in absencehistory table which doesn't have end date. Absence ends should be inserted only when date of end of absence is in the past. Each employee can have only one 'open' absence at any time.

2. Relationships among entities.


Entry in the training history table has to belong to one employee.
TRAINING HISTORY
Employee can attend many training courses.
PROMOTION HISTORY
EMPLOYEE
ABSENCE HISTORY
Each promotion have to be linked to one employee.
Employee can be absent many times.
Each absence has to be linked to one employee.
Employee has to have one or more promotions.

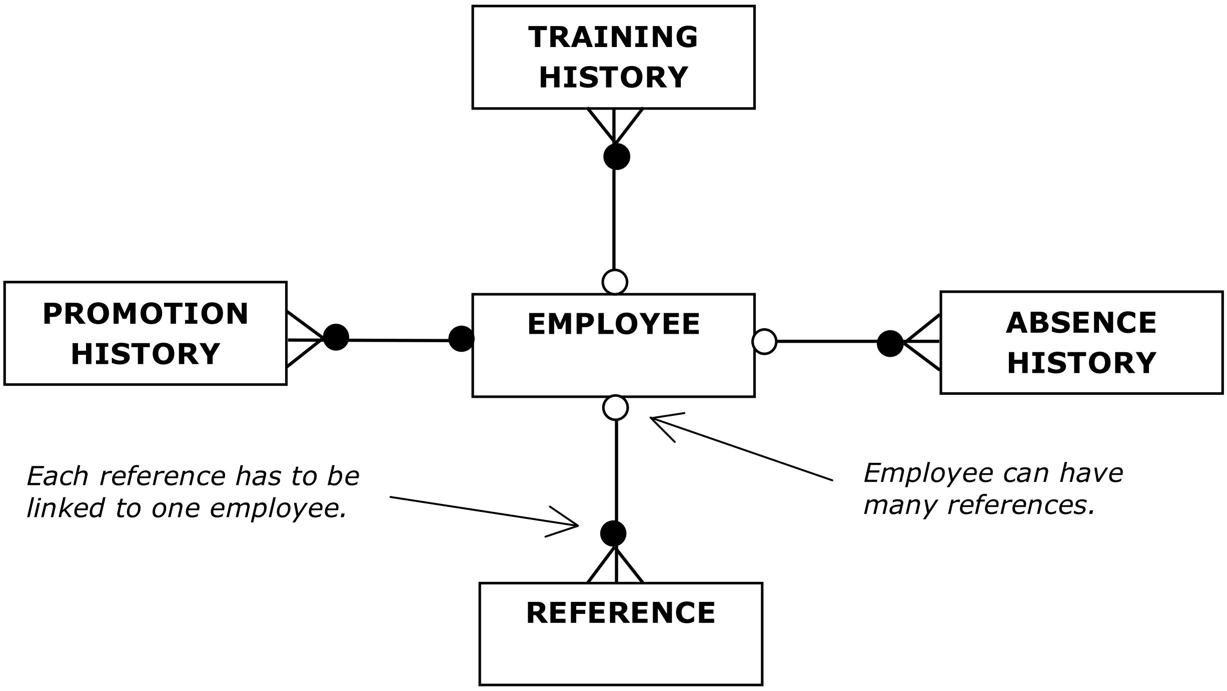
Namely the relationships between entities can be described as:

- Employee has to have one or more promotions (employment is considered a promotion).
- Each promotion has to be linked to exactly one employee (there is no point of promoting somebody who is not an employee).
- Employee can be absent 0, 1 or many times.
- Each absence has to be linked to exactly one employee.
- Employee can attend 0, 1 or many training courses.
- Each training course has to be linked to an employee (additionally training course has to have at least one employee attending it to be stored in the database).

Assuming that employee can provide many references creates a problem of internal structure or a list inside of employee table. This fact takes employee table out of 1NF.

To bring employee table back into the 1NF I need to create new table which will store all references provided by the employee during recruitment process.

Diagram below illustrates new entity relationship diagram with REFERENCE table (only new relationship is described on this diagram).


TRAINING HISTORY
PROMOTION HISTORY
EMPLOYEE
ABSENCE HISTORY
REFERENCE
Each reference has to be linked to one employee.
Employee can have many references.

Namely (relationship for the new entity):

- Employee can have 0, 1 or many references.
- Each reference has to be linked to exactly one employee.

Modified EMPLOYEE table and new table REFERENCE

EMPLOYEE table

Attribute Name	Description	Type
emp_id	Unique number identifying each employee.	varchar2 (4)
fname	First name of the employee.	varchar2 (15)
sname	Surname of the employee.	varchar2 (15)
DoB	Date of birth of the employee.	date
sex	Gender of the employee; it can take only values of 'M' (= male) or 'F' (=female).	char (1)
telephone	Telephone number of the employee.	number (11)
hname	Information concerned with the postal address (house number or name).	varchar2 (20)
street	Information concerned with the postal address.	varchar2 (20)
town	Information concerned with the postal address.	varchar2 (20)
postcode	Information concerned with the postal address.	varchar2 (8)
b_sortcode	Sort code of the bank account owned by the employee (often identifies branch of the bank)	number (6)
b_accno	Number of the account owned by the employee.	number (8)
b_name	Name of the bank of the employee.	varchar2 (15)
p_method	Preferred payment method (ie. cheque, bank transfer, cash) {'Q' | 'T' | 'C'}	char (1)

REFERENCE table

Attribute Name	Description	Type
emp_id	Unique number identifying each employee (foreign key).	varchar2 (4)
id	Unique id number identifying reference provided by the employee.	varchar2 (4)
name	Name of person referring this employee during the recruitment process.	varchar2 (30)
location	Location of that reference in the paper record.	varchar2 (4)

As id is uniquely identifying each reference in this table. Combination of emp_id and id creates a primary key for this table.

3. Representation of the model as a set of 1NF relations.

Following relationships are in 1NF, because:

- All the key attributes are defined.
- There is no repeating group.
- None of the attributes is a structure within table.
- All attributes depend on the key.

Primary keys are underlined and RED. All foreign keys are in *italic*.

EMPLOYEE (emp_id, fname, sname, DoB, sex, hname, street, town, postcode, b_sortcode, b_accno, b_name, p_method)

REFERENCE (*emp_id*, id, name, location)

PROMOTIONHISTORY (*emp_id*, prom_date, position, department, salary, mod, description, narrative)

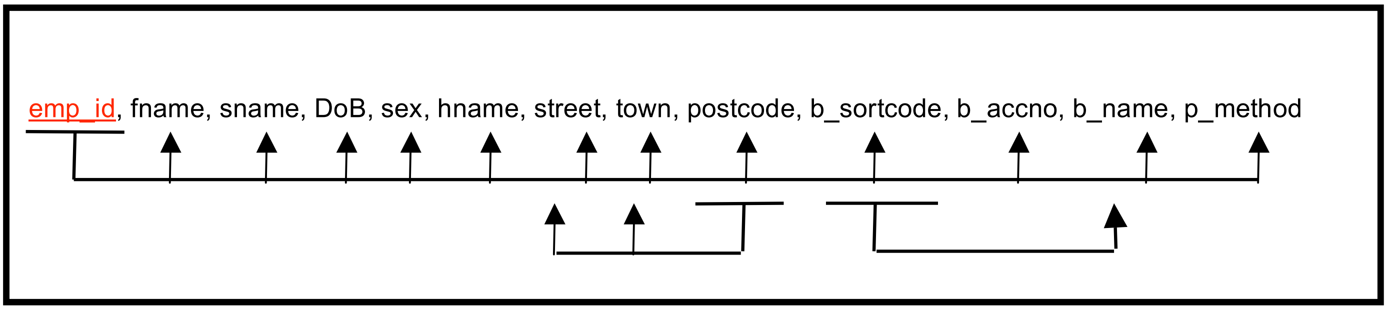
ABSENCEHISTORY (*emp_id*, abs_start, abs_end, code, reason, paid)

TRAININGHISTORY (*emp_id*, c_start, c_name, c_end, description, organiser, org_hnumber, org_street, org_town, org_postcode, location, result)

4. Functional dependencies and normalisation to BCNF.

4.1 EMPLOYEE table.

Employee table is already quite close to the BCNF. The functional dependencies look in the following way.


emp_id, fname, sname, DoB, sex, hname, street, town, postcode, b_sortcode, b_accno, b_name, p_method

Functional dependencies:

- All attributes depend on the primary key as whole.
- street depends on the postcode,
- town depends on the postcode,
- b_name (bank name) depends on b_sortcode,

It is worth noting that functional dependency of street and town on the postcode can be tricky to implement. Following principles of normalisation new tables for this information need to be created.

This means that before record for new employee is inserted there need to be a record in the POSTCODE table which would store the street name and town name.

This could be thought as unpractical, as multistage insertion is required. Benefit of that approach is that it minimises data redundancy (assuming that database is large).

Tables after normalisation

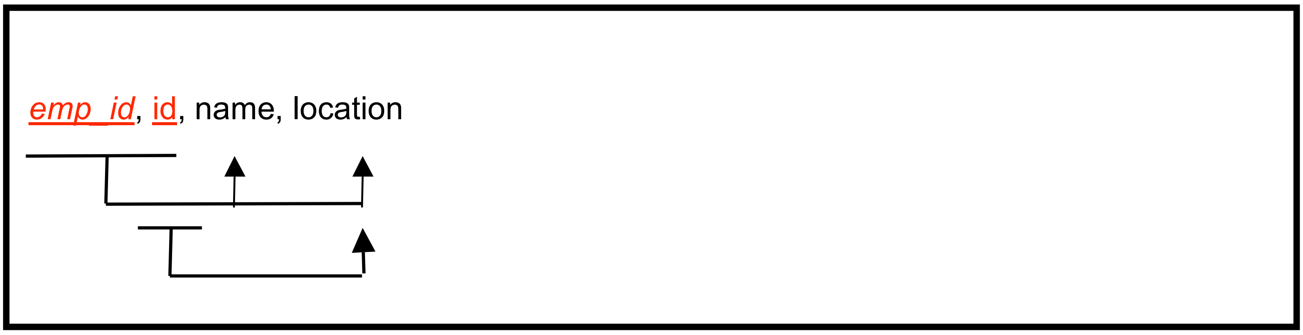
EMPLOYEE (emp_id, fname, sname, DoB, sex, hname, *postcode*, *sortcode*, accno, p_method)

POSTCODE (postcode, street, town)

BSORTCODE (sortcode, name)

4.2 REFERENCE table.

Reference table is very close to BCNF.



Functional dependencies:

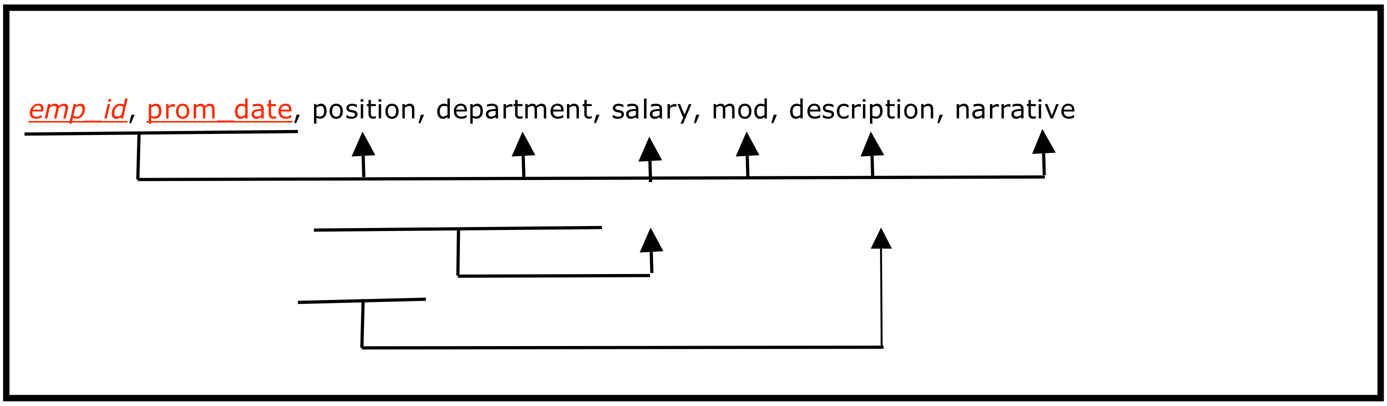
- All attributes depend on the primary key as a whole.
- Reference location depends on the id.

Tables after normalisation

REFINDEX (*emp_id*, *id*, name)

REFLOC (id, location)

4.3 PROMOTIONHISTORY table.


emp_id, prom_date, position, department, salary, mod, description, narrative

Functional dependencies:

- All attributes depend on the primary key as a whole.
- Salary depends on the position and the department.
- Description depends on the position.

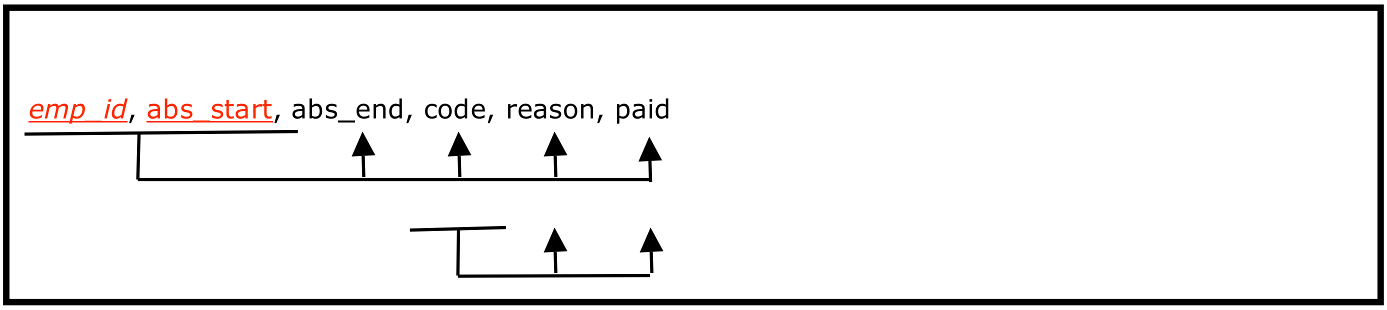
Tables after normalisation

PROMOTIONHISTORY (*emp_id*, prom_date, *position*, *department*, mod, narrative)

SALARY (position, department, salary)

DESCRIPTION (position, description)

4.4 ABSENCEHISTORY table.



emp_id, abs_start, abs_end, code, reason, paid

Functional dependencies:

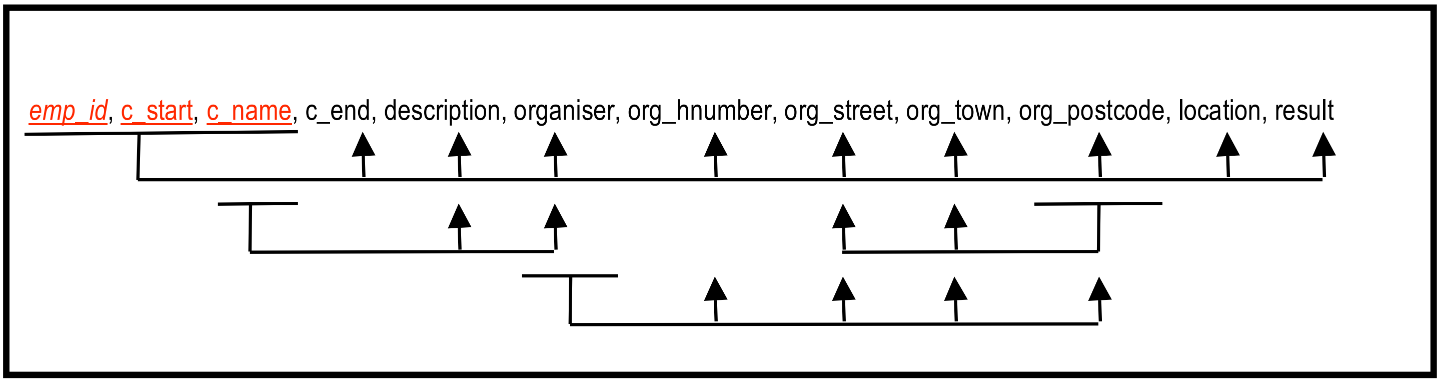
- All attributes depend on the primary key as a whole.
- Reason (describing the code) and paid depends on the code.

Tables after normalisation

ABSENCEHISTORY (*emp_id*, abs_start, abs_end, *code*)

REASON (code, reason, paid)

4.5 TRAININGHISTORY table.


emp_id, c_start, c_name, c_end, description, organiser, org_hnumber, org_street, org_town, org_postcode, location, result

Functional dependencies:

- All attributes depend on the primary key as a whole.
- Course description and course organiser depend on the course name.
- org_street and org_town depend on the postcode. It is worth noting that employee table has the same functional dependency. Table created during that normalisation should be used.
- Details of the organiser depend on that organiser's name.

Tables after normalisation

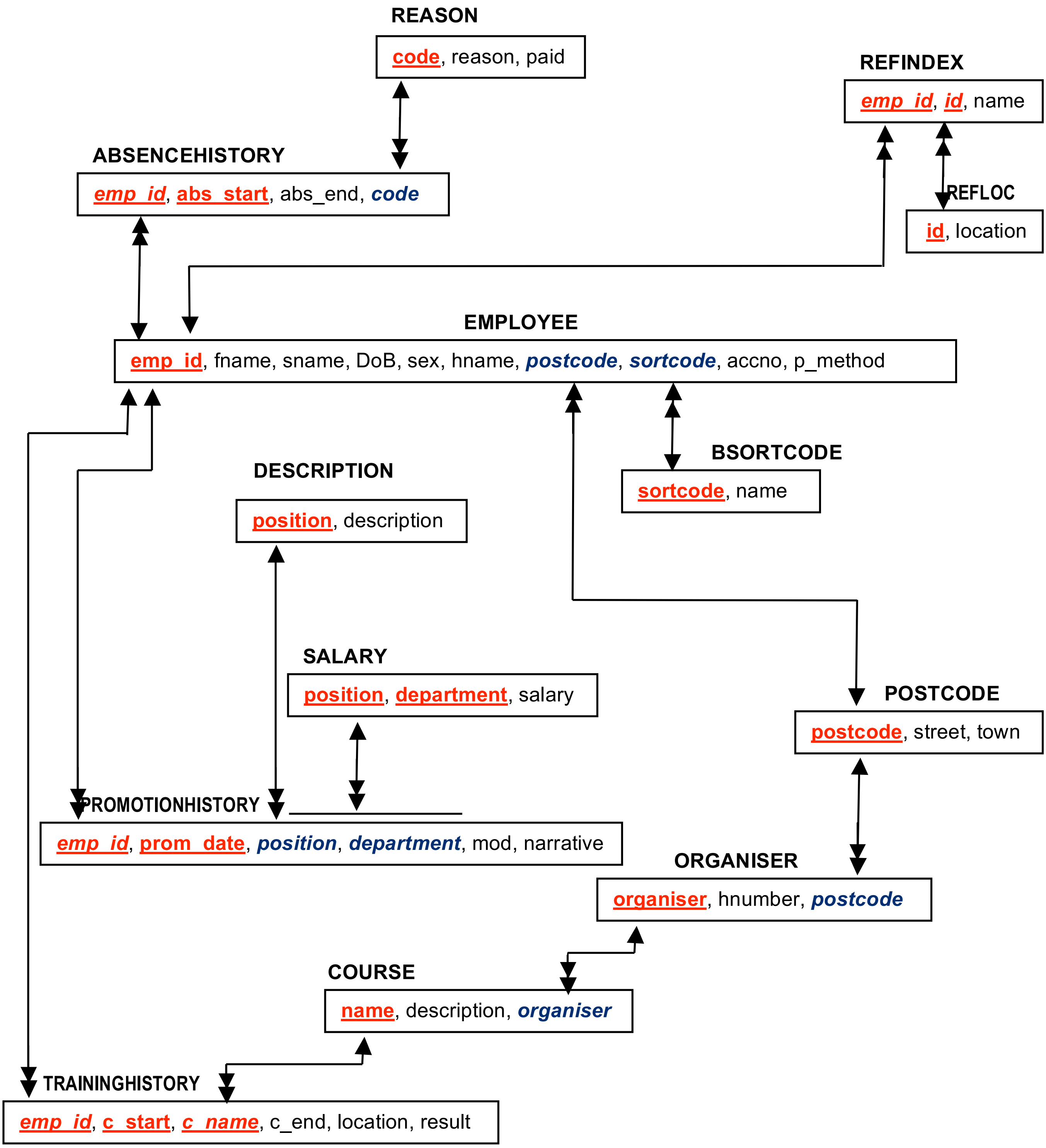
TRAININGHISTORY (*emp_id*, c_start, *c_name*, c_end, location, result)

COURSE (name, description, *organiser*)

ORGANISER (organiser, hnumber, *postcode*)

POSTCODE table created during normalisation of the EMPLOYEE table.

5. Referential integrity diagram.


REASON
code, reason, paid
REFINDEX
emp_id, id, name
ABSENCEHISTORY
emp_id, abs_start, abs_end, code
REFLOC
id, location
EMPLOYEE
emp_id, fname, sname, DoB, sex, hname, postcode, sortcode, accno, p_method
BSORTCODE
sortcode, name
DESCRIPTION
position, description
SALARY
position, department, salary
POSTCODE
postcode, street, town
PROMOTIONHISTORY
emp_id, prom_date, position, department, mod, narrative
ORGANISER
organiser, hnumber, postcode
COURSE
name, description, organiser
TRAININGHISTORY
emp_id, c_start, c_name, c_end, location, result

6. Definition of relationships in SQL

Script recording creation of all those tables can be found in the Appendix 1.

6.1 Creating table employee.

```
/* Creates EMPLOYEE table according to the specification below:
emp_id          - varchar2 (4)  - primary key
fname   - varchar2 (15) - not null
sname   - varchar2 (15) - not null
dob     - date          - not null      - greater than today-130 years
sex     - char (1)      - not null      - has to be 'M' or 'F'
hname   - varchar2 (20)- not null
postcode- varchar2 (8)- not null        - foreign key for POSTCODE
                                                table
sortcode- number (6)                    - foreign key for BSORTCODE
                                                table
accno   - number (8)
p_method- char (1)      - not null      - {'Q' | 'T' | 'C'}
*/

create table EMPLOYEE
(EMP_ID         varchar2(4) primary key,
 FNAME varchar2(15) not null,
 SNAME varchar2(15) not null,
 DOB date not null,
 SEX char(1) constraint a_sex_value check (SEX in ('M', 'F')),
 HNAME varchar2(20) not null,
 POSTCODE varchar2(8),
 SORTCODE number(6),
 ACCNO number(8),
 P_METHOD char(1) constraint a_p_method_value check (P_METHOD in
('Q', 'T', 'C')),
 constraint a_fkey_1 foreign key (POSTCODE) references
POSTCODE(POSTCODE) on delete set null,
 constraint a_fkey_2 foreign key (SORTCODE) references
BSORTCODE(SORTCODE) on delete set null);
```

6.2 Creating table postcode.

```
/* Creates POSTCODE table according to the specification */
create table POSTCODE
(POSTCODE varchar2(8) primary key,
 STREET varchar2(20),
 TOWN varchar2(20) not null);
```

6.3 Creating table bsortcode.

```
/* Creates BSORTCODE table according to the specification. */
create table BSORTCODE
(SORTCODE number(6) primary key,
 NAME varchar2(15) not null);
```

6.4 Creating table refloc.

```
/* Creates REFLOC table according to the specification. */

create table REFLOC
(ID varchar2(4) primary key,
 LOCATION varchar2(4) not null);
```

6.5 Creating table refindex.

```
/* Creates REFINDEX table according to the specification. */

create table REFINDEX
(EMP_ID varchar2(4),
 ID varchar2(4),
 NAME varchar2(30) not null,
 constraint a_skey_1 primary key (EMP_ID, ID),
 constraint a_fkey_3 foreign key (EMP_ID) references EMPLOYEE(EMP_ID),
 constraint a_fkey_4 foreign key (ID) references REFLOC(ID) ON DELETE CASCADE);
```

6.6 Creating table reason.

```
/* Creates REASON table according to the specification. */

create table REASON
(CODE varchar2(2) primary key,
 REASON varchar2(30) not null,
 PAID char(1) constraint a_paid_value check (PAID in ('Y', 'N')));
```

6.7 Creating table absencehistory.

```
/* Creates ABSENCEHISTORY table according to the specification. */

create table ABSENCEHISTORY
(EMP_ID varchar2(4),
 ABS_START date,
 ABS_END date,
 CODE varchar2(2) not null,
 constraint a_skey_4 primary key (EMP_ID, ABS_START),
 constraint a_fkey_8 foreign key (EMP_ID) references EMPLOYEE(EMP_ID),
 constraint a_fkey_9 foreign key (CODE) references REASON(CODE) initially deferred deferrable,
 constraint a_abs_value check (ABS_START<ABS_END));
```

6.8 Creating table description.

```
/* Creates DESCRIPTION table according to the specification. */

create table DESCRIPTION
(POSITION varchar2(20) primary key,
 DESCRIPTION varchar2(30) not null);
```

6.9 Creating table salary.

```
/* Creates SALARY table according to the specification. */

create table SALARY
(POSITION varchar2(20),
 DEPARTMENT varchar2(20),
 SALARY number(6) check (SALARY between 0 and 999999),
 constraint a_skey_2 primary key (POSITION, DEPARTMENT));
```

6.10 Creating table promotionhistory.

```
/* Creates PROMOTIONHISTORY table according to the specification. */

create table PROMOTIONHISTORY
(EMP_ID varchar2(4),
 PROM_DATE date,
 POSITION varchar2(20),
 DEPARTMENT varchar(20),
 MOD number(6),
 NARRATIVE varchar2(30),
 constraint a_skey_3 primary key (EMP_ID, PROM_DATE),
 constraint a_fkey_5 foreign key (EMP_ID) references EMPLOYEE(EMP_ID),
 constraint a_fkey_6 foreign key (POSITION, DEPARTMENT) references SALARY(POSITION,
DEPARTMENT) on delete set null,
 constraint a_fkey_7 foreign key (POSITION) references DESCRIPTION(POSITION) on delete set null);
```

6.11 Creating table organiser.

```
/* Creates ORGANISER table according to the specification. */

create table ORGANISER
(ORGANISER varchar2(20) primary key,
 HNUMBER varchar2(20) not null,
 POSTCODE varchar2(8),
 constraint a_fkey_11 foreign key (POSTCODE) references POSTCODE(POSTCODE) on delete set null);
```

6.12 Creating table course.

```
/* Creates COURSE table according to the specification. */

create table COURSE
(NAME varchar2(30) primary key,
 DESCRIPTION varchar2(30),
 ORGANISER varchar2(20),
 constraint a_fkey_10 foreign key (ORGANISER) references ORGANISER(ORGANISER) on delete set null);
```

6.13 Creating table traininghistory.

```
/* Creates TRAININGHISTORY table according to the specification. */

create table TRAININGHISTORY
(EMP_ID varchar2(4),
 C_START date,
 C_NAME varchar2(30),
 C_END date not null,
 LOCATION varchar2(50) not null,
 RESULT varchar2(30),
 constraint a_skey_5 primary key (EMP_ID, C_START, C_NAME),
 constraint a_fkey_12 foreign key (EMP_ID) references EMPLOYEE(EMP_ID),
 constraint a_fkey_13 foreign key (C_NAME) references COURSE(NAME) on delete set null);
```

7. Insertion transactions.

Please note that those transactions by themselves cannot assure referential integrity of the database and should be used with extreme caution.

To ensure that database stays intact pre conditions have to be satisfied.

Any number of those transactions could be combined into one transaction to enable easier insertion of the records (it can be done by simply pasting them into one .sql file). Due to very big number of possible combination only one example transaction is present in this document (insertion-of-employee).

Example runs of each transaction can be found in Appendix 2.

Example dataset (in sqlloader scripts) can be found in Appendix 3.

7.1 Absence insertion.

```
/*
PROCESS:insert-new-absence (emp_id,abs_start,abs_end,code)
FILE:   absence.sql
PRE:    (*) tables ABSENCEHISTORY, REASON, EMPLOYEE exist and are
          relationally consistent
        (*) absence is inserted for an employee existing in the
          EMPLOYEE table
        (*) code of the absence reason inserted is present in the
          REASON table
POST:   new record is inserted into ABSENCEHISTORY

IN:     &emp<--emp_id, &&ast<--abs_start, &&aed<--abs_end,
          &cod<--code

OUT:    {success | failure} message

LOGIC:  Simple insertion given that pre conditions are satisfied
        (additional conditions are enforced by able definition)

SQL:
*/

set verify off
prompt WARNING! Single table insert - use with caution!
prompt Remember about referencial integrity and constraints
prompt INSERTING NEW ABSENCE HISTORY ENTRY

prompt

accept emp prompt 'Emp_id of absent employee > '
accept ast prompt 'Start date of the absecence (DD-MM-YYYY) > '
accept aed prompt 'End date of the absence (DD-MM-YYYY) > '
accept cod prompt 'Absence reason > '

insert into ABSENCEHISTORY
select
'&&emp',to_date('&&ast','DD-MM-YYYY'),to_date('&&aed','DD-MM-YYYY'),'&cod'
from dual
where (to_date('&&ast','DD-MM-YYYY') < to_date('&&aed','DD-MM-YYYY')
or to_date('&&aed','DD-MM-YYYY') is null) and not exists (select *
from ABSENCEHISTORY where EMP_ID='&&emp' and ABS_END is null);

undefine emp;
undefine ast;
undefine aed;

commit;
```

7.2 Bsortcode insertion.

```
/*
PROCESS:        insert-new-sortcode (sortcode,name)
FILE:           bsortcode.sql
PRE:            table BSORTCODE exists and is relationally consistent
POST:           new record inserted into BSORTCODE

IN:             &sor<--sortcode, &nam<--name

OUT:            {success | failure} message

LOGIC:                  simple insertion which is not subject to many
                constraint apart from uniqness of the sortcode (which is enforced by
                the table definition)

SQL:                                                                */

set verify off

prompt WARNING! Single table insert - use with caution!
prompt Remember about referencial integrity and constraints
prompt INSERTING NEW BANK SORTCODE INTO THE DATABASE

accept sor prompt 'Bank`s sortcode > '
accept nam prompt 'Bank`s name > '

insert into BSORTCODE values (&sor,'&nam');
```

7.3 Course insertion.

```
/*
PROCESS:insert-new-course
FILE:   course.sql
PRE:    (*) tables COURSE and ORGANISER exist and are
        relationally consistent
     (*) course is inserted for an organiser existing in the
        ORGANISER table
POST:   new record is inserted into COURSE

IN:     &nam<--name, &des<--description, &org<--organiser

OUT:    {success | failure} message

LOGIC:  Simple insertion given that pre conditions are satisfied
     (additional conditions are enforced by able definition)

SQL:                                                        */

set verify off

prompt WARNING! Single table insert - use with caution!
prompt Remember about referencial integrity and constraints
prompt INSERTING NEW COURSE INTO DATABASE

prompt

accept nam prompt 'Name of the course > '
accept des prompt 'Description of this course > '
accept org prompt 'Organiser of the course > '

insert into COURSE values ('&nam','&des','&org');
```

7.4 Employee insertion

```
/*
PROCESS:insert-new-employee (according to the specification)
FILE:   employee.sql
PRE:    (*) emp_id of new employee is unique
        (*) tables EMPLOYEE,BSORTCODE,POSTCODE,PROMOTIONHISTORY,
            DESCRIPTION,SALARY exist and are rellationally consistent
        (*) table POSTCODE contains record which matches postcode of
            new employee
        (*) table BSORTCODE contains record which matches sortcode of
            bank of new employee
        (*) table DESCRIPTION contains record which matches position
            to which new employee is going to be promoted
        (*) table SALARY contains record which matches position and
            department of the new employee
POST:   new record inserted into EMPLOYEE and PROMOTIONHISTORY tables

IN:     &&emp<--emp_id, &fna<--fname, &sna<--sname, &dob<--DoB,
        &sex<--sex, &hna<--hname, &pos<--postcode, &sor<--sortcode,
        &acc<--accno, &pme<--p_method
        &prom<--prom_date, &pot<--position, &dep<--department,
        &mod<--mod, &nar<--narrative

LOGIC:          Simple insertion into the EMPLOYEE and POSITIONHISTORY table.
        All necessary conditions are specified in the PRE section
        and as long as all of them are true this insertion is very
        straight forward.
        During all of those multiple insertion referencial integrity
        is enforced by checks in this transaction or through table
        definitions.

SQL:                                                                    */

set autocommit off
set verify off

prompt WARNING! Single table insert - use with caution!
prompt Remember about referencial integrity and constraints
prompt INSERTION OF NEW EMPLOYEE AND PROMOTING HIM/HER TO RIGHT POSITION

accept emp prompt  'Emp_id of new employee > '
accept fna prompt  'First name of new employee > '
accept sna prompt  'Surname of new employee > '
accept dob prompt  'Date of birth of new employee (dd-mm-yyyy) > '
accept sex prompt  'Sex of new employee (M or F) > '
accept hna prompt  'House name or number of new employee > '
accept pos prompt  'Postcode of new employee (XXXX XXX format) > '
accept sor prompt  'Sortcode of bank of employee > '
accept acc prompt  'Account number of new employee > '
accept pme prompt  'Preffered payment method (Q or T or C) > '
accept pda prompt  'Date of the employment > '
accept pot prompt  'Position of new employee > '
accept dep prompt  'Department of new employee > '
accept mod prompt  'Base salary modification (optional) > '
accept nar prompt  'Comments regarding employment > '
```

EMPLOYEE INSERTION CONTINUED...

```
insert into EMPLOYEE
select '&&emp','&fna','&sna',to_date('&&dob','DD-MM-YYYY'),
       '&sex','&hna','&pos',&sor,&acc,'&pme' from dual
where (to_date('&&dob','DD-MM-YYYY') + to_yminterval('130-00')) >=
sysdate;

insert into PROMOTIONHISTORY values (
'&&emp',to_date('&pda','DD-MM-YYYY'),'&pot','&dep',&mod,'&nar'
);

undefine emp;
undefine dob;

set autocommit on;

commit;
```

7.5 Organiser insertion.

```
/*
PROCESS:insert-new-organiser (oraganiser,hnumber,postcode)
FILE:   organiser.sql
PRE:    (*) tables ORGANISER and POSTCODE exist and are
            relationally consistent
        (*) postcode of the new organiser already exists in the
            POSTCODE table
POST:   new record is inserted into ORGANISER

IN:     &org<--organiser, &hnu<--hnumber, &pos<--postcode

OUT:    {success | failure} message

LOGIC:  Simple insertion given that pre conditions are satisfied
        (additional conditions are enforced by able definition)

SQL:                                                            */

set verify off

prompt WARNING! Single table insert - use with caution!
prompt Remember about referencial integrity and constraints
prompt INSERTING NEW ORGANISER INTO THE DATABASE

prompt

accept org prompt 'Name of new organiser > '
accept hnu prompt 'House name or number of new organiser > '
accept pos prompt 'Postcode of new organiser (XXXX XXX format) > '

insert into ORGANISER values ('&org','&hnu','&pos');
```

7.6 Position insertion.

```
/*
PROCESS:      insert-new-position (position,description)
FILE:         insert-new-position.sql
PRE:          table DESCRIPTION exists and is relationally
              consistent
POST:         new record inserted into DESCRIPTION

IN:           &pos<--position, &desc<--description

OUT:          {success | failure} message

LOGIC:                simple insertion which is not subject to many
              constraints apart from uniqness of the position (which is enforced by
              the table definition)

SQL:                                                          */

set verify off

prompt WARNING! Single table insert - use with caution!
prompt Remember about referencial integrity and constraints
prompt INSERTING NEW POSITION INTO THE DATABASE

accept pos prompt 'Position to be inserted: '
accept desc prompt 'Description of the position: '

insert into DESCRIPTION values
('&pos','&desc');
```

7.7 Postcode insertion.

```
/*
PROCESS:    insert-new-postcode (postcode,street,town)
FILE:       postcode.sql
PRE:        table POSTCODE exists and is relationally consistent
POST:       new record inserted into POSTCODE

IN:         &&pos<--postcode, &str<--street, &tow<--town

OUT:        {success | failure} message

LOGIC:              simple insertion which is not subject to many
            constraints apart from uniqness of the postcode (which is enforced by
            the table definition)

SQL:                                                        */

set verify off

prompt WARNING! Single table insert - use with caution!
prompt Remember about referencial integrity and constraints
prompt INSERTING NEW POSTCODE INTO THE DATABASE

accept pos prompt 'Postcode to be added (XXXX XXX format): '
accept str prompt 'Street: '
accept tow prompt 'Town: '

insert into POSTCODE values
('&pos','&str','&tow');
```

7.8 Promotion insertion.

```
/*
PROCESS:promote-employee
FILE:   promotion.sql
PRE:    (*) tables PROMOTIONHISTORY, DESCRIPTIO, POSITION,
            EMPLOYEE exist and are relationally consistent
      (*) prmotion is inserted for an employee existing in the
          EMPLOYEE table
      (*) employee is pormoted to position existing in the
            description table
          (*) employee is promoted to combination of position and
            department existing in the database
POST:   new record is inserted into PROMOTIONHISTORY

IN:     &emp<--emp_id, &pda<--prom_date, &pos<--position,
          &dep<--department, &mod<--mod, &nar<--narrative

OUT:    {success | failure} message

LOGIC:  Simple insertion given that pre conditions are satisfied
      (additional conditions are enforced by able definition).
          Additionally &pda has to be greater than any other prom_date
          for that employee.

SQL:                                                            */

prompt WARNING! Single table insert - use with caution!
prompt Remember about referencial integrity and constraints
prompt PROMOTING EXISTING EMPLOYEE

prompt

accept emp prompt 'Emp_id of promoted/demoted employee > '
accept pda prompt 'Date of promotion/demotion > '
accept pos prompt 'New position > '
accept dep prompt 'New department > '
accept mod prompt 'Basic salary modification > '
accept nar prompt 'Optional comments > '

set verify off

insert into PROMOTIONHISTORY
select '&&emp',to_date('&&pda','DD-MM-YYYY'),'&pos','&dep',&mod,'&nar'
          from dual
where to_date('&&pda','DD-MM-YYYY')> all (select PROM_DATE from
PROMOTIONHISTORY where EMP_ID='&&emp');


undefine emp;
undefine pda;
```

7.9 Absence reason insertion.

```
/*
PROCESS:insert-new-absence-reason (code,reason,paid)
FILE:   reason.sql
PRE:    table REASON exists
POST:   new records in REASON is inserted

IN:     &cod<--code, &res<--reason, &p<--paid

OUT:    {success | failure} message

LOGIC:          Simple insertion with no additinal constraints apart from
        ones enforced by the table definition.

SQL:                                                                    */

set verify off

prompt WARNING! Single table insert - use with caution!
prompt Remember about referencial integrity and constraints
prompt INSERTING NEW ABSENCE REASON

prompt

accept cod prompt 'Code of new absence reason > '
accept res prompt 'Description of this absence reason > '
accept p prompt 'Paid? (Y/N) > '

insert into REASON values ('&cod','&res','&p');
```

7.10 Reference insertion.

```
/*
PROCESS:insert-new-reference (emp_id,id,name,location)
FILE:   reference.sql
PRE:    (*) tables REFINDEX, REFLOC and EMPLOYEE exist and are
            relationally consistent
        (*) reference inserted is assigned to existing employee
POST:   new records in REFINDEX and REFLOC are inserted

IN:     &emp<--emp_id, &id<--id, &nam<--namem, &loc<--location

OUT:    {success | failure} message

LOGIC:          (1) record containg information regarding location of the
            reference is inserted first
        (2) infromation is inserted into REFINDEX table

SQL:                                                            */

set verify off

prompt WARNING! Single table insert - use with caution!
prompt Remember about referencial integrity and constraints
prompt INSERTING NEW REFERENCE INTO THE DATABASE

prompt

accept emp prompt 'Id of referenced employee > '
accept id prompt 'Id of the paper reference > '
accept loc prompt 'Location of the paper reference > '
accept nam prompt 'Name of referee > '

insert all
into REFLOC values ('&id','&loc')
into REFINDEX values ('&emp','&id','&nam')
select * from dual;
```

7.11 Salary insertion.

```
/*
PROCESS:        insert-new-salary (position,department,salary)
FILE:           insert-new-salary.sql
PRE:            table SALARY exists and is relationally consistent
POST:           new record inserted into SALARY

IN:             &pos<--position, &dep<--department, &sal<--salary

OUT:            {success | failure} message

LOGIC:                  simple insertion which is not subject to many
                constraints apart from uniqness of the (position,department) (which
                is enforced by the table definition)

SQL:                                                                    */

set verify off

prompt WARNING! Single table insert - use with caution!
prompt Remember about referencial integrity and constraints
prompt INSERTING NEW SALARY INTO THE DATABASE

accept pos prompt 'Position to be inserted: '
accept dep prompt 'Department: '
accept sal prompt 'Salary: '

insert into SALARY values
('&pos','&dep',&sal);
```

7.12 Training insertion.

```
/*
PROCESS:insert-new-training
FILE:   training.sql
PRE:    (*) tables TRAININGHISTORY, COURSE, ORGANISER, POSTCODE exist
            and are relationally consistent
        (*) training is inserted for an employee existing in the
            EMPLOYEE table
        (*) organiser of that training course is in the database
            (*) course for which employee is registering is in the
                database
POST:   new record is inserted into TRAININGHISTORY

IN:     &emp<--emp_id, &cst<--c_start, &cna<--c_name, &cen<--c_end,
          &loc<--location, &res<--result

OUT:    {success | failure} message

LOGIC:  Simple insertion given that pre conditions are satisfied
        (additional conditions are enforced by able definition)

SQL:                                                            */

set verify off

prompt WARNING! Single table insert - use with caution!
prompt Remember about referencial integrity and constraints
prompt INSERTING NEW TRAININGHISTORY ENTRY INTO THE DATABASE

prompt

accept emp prompt 'Emp_id of the employee > '
accept cst prompt 'Date of start of a course (DD-MM-YYYY) > '
accept cna prompt 'Name of the course > '
accept cen prompt 'Date of end of a course (DD-MM-YYYY) > '
accept loc prompt 'Location of the trainign course > '
accept res prompt 'Result of training course > '

insert into TRAININGHISTORY
select '&&emp',to_date('&&cst','DD-MM-YYYY'),'&&cna',
to_date('&cen','DD-MM-YYYY'),'&loc','&res'
from dual
where to_date('&&cst','DD-MM-YYYY') < to_date('&&ced','DD-MM-YYYY');

undefine emp
undefine cst
undefine cna
```

8. Delete-employee transaction.

8.1 Specification and logic.

PROCESS:delete-employee

FILE: delete-employee.sql
PRE: all tables specified in the documentation exist and
are relationally consistent.
POST: Record of the selected employee is deleted including all
related redundant data (employees promotions, etc.)

IN: &&emp<--emp_id

OUT: Rolling deletion reports in form <<Table affected>>
<<number of rows deleted>>.
In case of specified emp_id not present in the database no
rows will be deleted.
It is worth noting that if employee exists at least 2 rows
will be deleted.

LOGIC: Quite complicate as the deletion propagates across whole database. Every table is affected and most of them will have to have some rows deleted. Deletion query should follow this outline (order to some extent is optional):

(1) Delete all references associated with this employee (simple deletion of all rows in REFLOC which have ID in list of ids in REFINDEX - on delete cascade).

(2) Delete sortcode of the employee from BSORTCODE table, if no other employee shares the same sortcode.

(3) Delete postcode of the employee from POSTCODE table, as long as there is not other employee or organiser having the same postcode.

(4) Delete absence reasons which are used only by deleted employee.

(5) Delete ABSENCEHISTORY records of the employee.

(6) Delete position descriptions which are used only by
employee to be deleted.

(7) Delete salaries records which are used only by employee to
be deleted.

(8) Delete PROMOTIONHISTORY records of the employee.

(9) Delete other unused postcodes.

(10) Delete organisers of courses attended only by employee
to be deleted.

(11) Delete courses which were only attended by employee to
be deleted (as course name is part of the primary key of
TRAININGHISTORY table, a_skey_5 constraint has to be temporarily
disabled).

(12) Delete TRAININGHISTORY records assigned to employee to
be deleted.

(13) Delete entry of employee to be deleted from EMPLOYEE
table.

8.2 SQL code.

Prompts are highlighted in green.

```
set autocommit off;
set verify off;

prompt ............ DELETING EMPLOYEE ............
prompt

accept emp prompt 'Emp_id of employee to be deleted > '

prompt Deleting References assigned to that employee

delete from REFLOC where ID in (select ID from REFINDEX where
EMP_ID='&&emp');

prompt Deleting bank record of the employee

delete from BSORTCODE where SORTCODE=(select SORTCODE from EMPLOYEE
where EMP_ID='&&emp') and not exists (select * from EMPLOYEE where
SORTCODE=(select SORTCODE from EMPLOYEE where EMP_ID='&&emp') and
EMP_ID!='&&emp');

prompt Deleting postcode record of the employee

delete from POSTCODE where POSTCODE=(select POSTCODE from EMPLOYEE
where EMP_ID='&&emp') and not exists (select * from EMPLOYEE where
POSTCODE=(select POSTCODE from EMPLOYEE where EMP_ID='&&emp') and
EMP_ID!='&&emp') and not exists (select * from ORGANISER where
POSTCODE=(select POSTCODE from EMPLOYEE where EMP_ID='&&emp'));

prompt Deleting unused absence reasons

delete from REASON where CODE IN ((select CODE from ABSENCEHISTORY
where EMP_ID='&&emp') minus (select CODE from ABSENCEHISTORY where
EMP_ID!='&&emp'));

prompt Deleting ABSENCEHISTORY records of the employee

delete from ABSENCEHISTORY where EMP_ID='&&emp';
```

```
prompt Deleting unused position descriptions

delete from DESCRIPTION where POSITION in ((select POSITION from
PROMOTIONHISTORY where EMP_ID='&&emp') minus (select POSITION from
PROMOTIONHISTORY where EMP_ID!='&&emp'));

prompt Deleting unused salary entries

delete from SALARY where POSITION in ((select POSITION from
PROMOTIONHISTORY where EMP_ID='&&emp') minus (select POSITION from
PROMOTIONHISTORY where EMP_ID!='&&emp')) and DEPARTMENT in ((select
DEPARTMENT from PROMOTIONHISTORY where EMP_ID='&&emp') minus (select
DEPARTMENT from PROMOTIONHISTORY where EMP_ID!='&&emp'));

prompt Deleting PROMOTIONHISTORY record of the employee

delete from PROMOTIONHISTORY where EMP_ID='&&emp';

prompt Deleting unused postcodes

delete from POSTCODE where POSTCODE in ((select distinct POSTCODE from
ORGANISER,COURSE,TRAININGHISTORY where
TRAININGHISTORY.C_NAME=COURSE.NAME and
COURSE.ORGANISER=ORGANISER.ORGANISER and
TRAININGHISTORY.EMP_ID='&&emp') minus (select distinct POSTCODE from
ORGANISER,COURSE,TRAININGHISTORY where
TRAININGHISTORY.C_NAME=COURSE.NAME and
COURSE.ORGANISER=ORGANISER.ORGANISER and
TRAININGHISTORY.EMP_ID!='&&emp'));

prompt Deleting unused organisers

delete from ORGANISER where ORGANISER in ((select distinct ORGANISER from
TRAININGHISTORY,COURSE where TRAININGHISTORY.C_NAME=COURSE.NAME and
TRAININGHISTORY.EMP_ID='&&emp') minus (select distinct ORGANISER from
TRAININGHISTORY,COURSE where TRAININGHISTORY.C_NAME=COURSE.NAME and
TRAININGHISTORY.EMP_ID!='&&emp'));

/* To enable this deletion to be performed temporarily constraint
a_skey_5 (primary key specification) has to be dropped */

prompt Deleting unused courses
```

```
alter table TRAININGHISTORY disable constraint a_skey_5;

delete from COURSE where NAME in ((select distinct C_NAME from
TRAININGHISTORY where EMP_ID='&&emp') minus
(select distinct C_NAME from TRAININGHISTORY where EMP_ID!='&&emp'));

alter table TRAININGHISTORY enable novalidate constraint a_skey_5;

prompt Deleting TRAININGHISTORY records of the employee

delete from TRAININGHISTORY where EMP_ID='&&emp';

prompt Deleting EMPLOYEE records of the employee

delete from EMPLOYEE where EMP_ID='&&emp';

set verify on;
commit;
```

8.3 Illustrative test run.

Script containing illustrative test run can be found in Appendix 4.

From the test run I can conclude that transaction is working as expected and only required data is deleted.

9. Query – most absent employee.

9.1 Specification.

```
PROCESS:most-absent-employee
FILE:   most-absent.sql
PRE:    all tables specified in the documentation exist and are
        relationally consistent
POST: true

IN:     no input

OUT:   details of most frequently absent employee are printed on the
       screen (most of absencehisotry entries found)
       If there are more than one employee with the same number of
       absences, records of all those employees will be printed.

LOGIC:          S1: employees with most absences
        S1:=max(card){x,card(ImageSet(emp_id=x)) forall x=project
        ABSENCEHISTORY (emp_id)}

        S2: data stored in the employee table of employee with most
        absences
        S2:=select EMPLOYEE (emp_id IN S1)
```

9.2 SQL code.

```
select * from EMPLOYEE
where EMP_ID IN (select EMP_ID from ABSENCEHISTORY
group by EMP_ID
having count(*)>= all (select count(*) from ABSENCEHISTORY group by
EMP_ID));
```

9.3 Illustrative test run.

Content of table ABSENCEHISTORY.

```
SQL> select * from absencehistory;

EMP_ ABS_START ABS_END   CO
---- --------- --------- --
T2   02-DEC-01 05-DEC-01 A2
T2   05-JAN-02 07-JAN-02 A1
T2   16-MAY-01 17-MAY-01 B1
T5   01-DEC-01 10-DEC-01 A2
T5   03-JAN-02 05-JAN-02 A2
T5   12-JAN-02 13-JAN-02 B1
T2   01-MAY-02 02-MAY-02 B1
```

Result of running most-absent.sql

```
SQL> @most-absent


EMP_ FNAME           SNAME           DOB       S HNAME                POSTCODE SORTCODE   ACCNO      P
---- --------------- --------------- --------- - -------------------- -------- ---------- ---------- -
T2   Mr.             Brown           12-FEB-72 M 34                   BR2 0DW  121212     11112222   C
```

Clearly result is as expected.

Appendix 1 – Table Creation Record

```
cs3% script record1.txt
Script started, file is record1.txt
cs3% sqlplus

SQL*Plus: Release 10.2.0.2.0 - Production on Tue Apr 15 16:35:20 2008

Copyright (c) 1982, 2005, Oracle.  All Rights Reserved.


Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.2.0 - 64bit Production
With the Partitioning, OLAP and Data Mining options

SQL> @createBsortcode

Table created.

SQL> desc bsortcode
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 SORTCODE                                  NOT NULL NUMBER(6)
 NAME                                      NOT NULL VARCHAR2(15)

SQL> @createPostcode

Table created.

SQL> desc postcode
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 POSTCODE                                  NOT NULL VARCHAR2(8)
 STREET                                             VARCHAR2(20)
 TOWN                                      NOT NULL VARCHAR2(20)

SQL> @createRefLoc

Table created.

SQL> desc refloc
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ID                                        NOT NULL VARCHAR2(4)
 LOCATION                                  NOT NULL VARCHAR2(4)
```

```
SQL> @createRefIndex

Table created.

SQL> desc refindex
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMP_ID                                    NOT NULL VARCHAR2(4)
 ID                                        NOT NULL VARCHAR2(4)
 NAME                                      NOT NULL VARCHAR2(30)

SQL> @createEmployee

Table created.

SQL> desc employee
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMP_ID                                    NOT NULL VARCHAR2(4)
 FNAME                                     NOT NULL VARCHAR2(15)
 SNAME                                     NOT NULL VARCHAR2(15)
 DOB                                       NOT NULL DATE
 SEX                                                CHAR(1)
 HNAME                                     NOT NULL VARCHAR2(20)
 POSTCODE                                           VARCHAR2(8)
 SORTCODE                                           NUMBER(6)
 ACCNO                                              NUMBER(8)
 P_METHOD                                           CHAR(1)

SQL> @createReason

Table created.

SQL> desc reason
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CODE                                      NOT NULL VARCHAR2(2)
 REASON                                    NOT NULL VARCHAR2(30)
 PAID                                               CHAR(1)

SQL> @createAbsenceHistory

Table created.

SQL> desc absencehistory
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMP_ID                                    NOT NULL VARCHAR2(4)
 ABS_START                                 NOT NULL DATE
 ABS_END                                            DATE
 CODE                                      NOT NULL VARCHAR2(2)

SQL> @createDescription
```

```
Table created.

SQL> desc description
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 POSITION                                  NOT NULL VARCHAR2(20)
 DESCRIPTION                               NOT NULL VARCHAR2(30)

SQL> @createSalary

Table created.

SQL> desc salary
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 POSITION                                  NOT NULL VARCHAR2(20)
 DEPARTMENT                                NOT NULL VARCHAR2(20)
 SALARY                                             NUMBER(6)

SQL> @createPromotionHistory

Table created.

SQL> desc promotionhistory
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMP_ID                                    NOT NULL VARCHAR2(4)
 PROM_DATE                                 NOT NULL DATE
 POSITION                                           VARCHAR2(20)
 DEPARTMENT                                         VARCHAR2(20)
 MOD                                                NUMBER(6)
 NARRATIVE                                          VARCHAR2(30)

SQL> @createOrganiser

Table created.

SQL> desc organiser
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ORGANISER                                 NOT NULL VARCHAR2(20)
 HNUMBER                                   NOT NULL VARCHAR2(20)
 POSTCODE                                           VARCHAR2(8)

SQL> @createCourse

Table created.

SQL> desc course
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 NAME                                      NOT NULL VARCHAR2(30)
 DESCRIPTION                                        VARCHAR2(30)
 ORGANISER                                   VARCHAR2(20)

SQL> @createTrainingHistory

Table created.

SQL> desc trainingHistory
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMP_ID                                    NOT NULL VARCHAR2(4)
 C_START                                   NOT NULL DATE
 C_NAME                                    NOT NULL VARCHAR2(30)
 C_END                                     NOT NULL DATE
 LOCATION                                  NOT NULL VARCHAR2(50)
 RESULT                                             VARCHAR2(30)

SQL> quit
Disconnected from Oracle Database 10g Enterprise Edition Release 10.2.0.2.0 - 64bit Production
With the Partitioning, OLAP and Data Mining options
cs3% exit
```

Appendix 2 – Example inserts

```
Script started on Tue Apr 15 20:46:47 2008
cs3% sqlplus

SQL*Plus: Release 10.2.0.2.0 - Production on Tue Apr 15 20:46:51 2008

Copyright (c) 1982, 2005, Oracle.  All Rights Reserved.


Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.2.0 - 64bit Production
With the Partitioning, OLAP and Data Mining options

SQL> @postcode
WARNING! Single table insert - use with caution!
Remember about referencial integrity and constraints
INSERTING NEW POSTCODE INTO THE DATABASE
Postcode to be added (XXXX XXX format): OX17 1RL
Street: Co row Lane
Town: Banbury

1 row created.

SQL> @bsortcode
WARNING! Single table insert - use with caution!
Remember about referencial integrity and constraints
INSERTING NEW BANK SORTCODE INTO THE DATABASE
Bank`s sortcode > 123456
Bank`s name > First Direct

1 row created.

SQL> @position
WARNING! Single table insert - use with caution!
Remember about referencial integrity and constraints
INSERTING NEW POSITION INTO THE DATABASE
Position to be inserted: Developer
Description of the position: Ordinary footsoldier

1 row created.

SQL> @salary
WARNING! Single table insert - use with caution!
Remember about referencial integrity and constraints
INSERTING NEW SALARY INTO THE DATABASE
Position to be inserted: Developer
Department: Java
Salary: 20000
```

```
1 row created.

SQL> @employee
WARNING! Single table insert - use with caution!
Remember about referencial integrity and constraints
INSERTION OF NEW EMPLOYEE AND PROMOTING HIM/HER TO RIGHT POSITION
Emp_id of new employee > A001
First name of new employee > Jakub
Surname of new employee > Deka
Date of birth of new employee (dd-mm-yyyy) > 30-08-1986
Sex of new employee (M or F) > M
House name or number of new employee > Little Barn House
Postcode of new employee (XXXX XXX format) > OX17 1RL
Sortcode of bank of employee > 123456
Account number of new employee > 12345678
Preffered payment method (Q or T or C) > T
Date of the employment > 13-04-2008
Position of new employee > De eveloper
Department of new employee > Java
Base salary modification (optional) > 0
Comments regarding employment > Test record

1 row created.


1 row created.


Commit complete.

SQL> @organiser
WARNING! Single table insert - use with caution!
Remember about referencial integrity and constraints
INSERTING NEW ORGANISER INTO THE DATABASE

Name of new organiser > Deka Training
House name or number of new organiser > Little Barn House
Postcode of new organiser (XXXX XXX format) > OX17 1RL

1 row created.

Commit complete.
SQL> @course
WARNING! Single table insert - use with caution!
Remember about referencial integrity and constraints
INSERTING NEW COURSE INTO DATABASE

Name of the course > Generics in Java
Description of this course > Basic course
Organiser of the course > Deka Training

1 row created.

Commit complete.
```

```
SQL> @training
WARNING! Single table insert - use with caution!
Remember about referencial integrity and constraints
INSERTING NEW TRAININGHISTORY ENTRY INTO THE DATABASE

Emp_id of the employee > A001
Date of start of a course (DD-MM-YYYY) > 14-04-2008
Name of the course > Generics in Java
Date of end of a course (DD-MM-YYYY) > 16-04-2008
Location of the trainign course > Bicester Training Office, Talisman Center
Result of training course > Pass
Enter value for ced: 16-04-2008

1 row created.

Commit complete.

SQL> @reference
WARNING! Single table insert - use with caution!
Remember about referencial integrity and constraints
INSERTING NEW REFERENCE INTO THE DATABASE

Id of referenced employee > A001
Id of the paper reference > A1
Location of the paper reference > B01
Name of referee > Ian McKenzie

2 rows created.

SQL> @absence
WARNING! Single table insert - use with caution!
Remember about referencial integrity and constraints
INSERTING NEW ABSENCE HISTORY ENTRY

Emp_id of absent employee > A001
Start date of the absecence (DD-MM-YYYY) > - 15-04-2008
End date of the absence (DD-MM-YYYY) > 16-04-2008
Absence reason > A1

1 row created.


Commit complete.

SQL> quit
Disconnected from Oracle Database 10g Enterprise Edition Release 10.2.0.2.0 - 64bit Production
With the Partitioning, OLAP and Data Mining options
cs3% exit
cs3%
script done on Tue Apr 15 21:16:48 2008
```

Absence table.

```
LOAD DATA
INFILE *
APPEND
INTO TABLE absencehistory
FIELDS TERMINATED BY "," OPTIONALLY ENCLOSED BY '"'
(EMP_ID, ABS_START DATE "DD-MM-YYYY", ABS_END DATE "DD-MM-YYYY",
CODE)
BEGINDATA
T1,15-05-2001,21-05-2001,A1
T2,16-05-2001,17-05-2001,B1
T5,01-12-2001,10-12-2001,A2
T6,01-11-2001,02-11-2001,B2
T6,08-11-2001,09-11-2001,B2
T6,15-11-2001,16-11-2001,B1
T1,16-12-2001,18-12-2001,A3
```

Bsortcode table.

```
LOAD DATA
INFILE *
APPEND
INTO TABLE BSORTCODE
FIELDS TERMINATED BY "," OPTIONALLY ENCLOSED BY '"'
(SORTCODE, NAME)
BEGINDATA
123456,A Bank
121212,B Bank
111777,First Direct
888888,HSBC
990133,Barlays
100007,Lloyds Premium
228743,Bank of China
```

Course table.

```
LOAD DATA
INFILE *
APPEND
INTO TABLE COURSE
FIELDS TERMINATED BY "," OPTIONALLY ENCLOSED BY '"'
(NAME, DESCRIPTION, ORGANISER)
BEGINDATA
Basic of C#,Course for Java programmers,C# Masters
Topics in C#,Reminder for C# programmers,C# Masters
Generics in C#,Advanced topic in genercis of C#,C# Masters
Design Patterns in C#,For most advanced C# programmers,C# Masters
Health and Safty in IT,General compulsory course,Helath and Safety
What is ethics in IT?,Lecture given by Mr. Brown,BCS
...build bridges,How to manage big IT projects,We know how to...
...work with people,Course covering motivation of the staff,We know how to...
Adv. topics in Java,only for senior developers,London IT
Adv. topics in .NET,only for senior developers,London IT
```

Description table.

```
LOAD DATA
INFILE *
APPEND
INTO TABLE DESCRIPTION
FIELDS TERMINATED BY "," OPTIONALLY ENCLOSED BY '"'
(POSITION, DESCRIPTION)
BEGINDATA
Janitor,Cleaning etc.
Secretary,Typing and filing
PA,pa-ing
Junior Developer,Minor development tasks
Developer,Work on bigger projects
Senior Developer,Lead big projects
Manager,Manage people
```

Employee table.

```
LOAD DATA
INFILE *
APPEND
INTO TABLE EMPLOYEE
FIELDS TERMINATED BY "," OPTIONALLY ENCLOSED BY '"'
(EMP_ID, FNAME, SNAME, DOB DATE "DD-MM-YYYY", SEX, HNAME, POSTCODE,
SORTCODE, ACCNO,
P_METHOD)
BEGINDATA
T1,Jakub,Deka,30-08-1986,M,Little Barn,NP4 5HY,123456,12121212,T
T2,Mr.,Brown,12-02-1972,M,34,BR2 0DW,121212,11112222,C
T3,Mrs.,Green,01-05-1987,F,The Manor,PL7 2ZN,100007,01010101,Q
T4,Roger,Sulivan,18-11-1976,M,11,WA9 1AG,228743,12345678,T
T5,Maria,Kevlarova,30-05-1956,F,Blue House,EX16 6PQ,111777,18181818,C
T6,Mike,Ballantine,17-12-1950,M,56,S62 6ES,990133,87654321,Q
```

Organiser table.

```
LOAD DATA
INFILE *
APPEND
INTO TABLE ORGANISER
FIELDS TERMINATED BY "," OPTIONALLY ENCLOSED BY '"'
(ORGANISER, HNUMBER, POSTCODE)
BEGINDATA
C# Masters,2a,NP4 5HY
Helath and Safety,67,NW10 2EA
IT Training Grou,IT House,BR2 0DW
BCS,BCS Place,GL20 7NT
We know how to...,9,EX16 6PQ
London IT,99,NW10 2EA
```

Postcode table.

```
LOAD DATA
INFILE *
APPEND
INTO TABLE POSTCODE
FIELDS TERMINATED BY "," OPTIONALLY ENCLOSED BY '"'
(POSTCODE, STREET, TOWN)
BEGINDATA
NP4 5HY,Windsor Road,Gwent
PA5 8JZ,High Street,Johnstone
NG18 2NS,Pecks Hill,Mansfield
DY8 4NH,High St,Stourbridge
WA11 9HB,Chain Lane,Merseyside
BB1 8DW,East Park Road,Lancashire
NW10 2EA,High Rd,London
GU17 0AB,London Rd,Camberley
B64 5HG,High Street,West Midlands
BR2 0DW,Beckenham Lane,Bromley
WV1 3EX,Dudley St,Wolverhampton
DN1 1HG,Silver St,Doncaster
BS4 2QB,Wells Rd,Bristol
WA9 1AG,Higher Parr Street,St. Helens
RH7 6EP,East Grinstead Rd,Lingfield
PL7 2ZN,St Stephens Place,Ridgeway
SW7 4SF,Gloucester Rd,London
S62 6ES,Broad St,Rotherham
CA7 9NJ,High St,Wigton
GL20 7NT,Overbury,Tewkesbury
BT42 3DH,Ballee Rd East,Ballymena
CH5 2LE,Parkway,Deeside
BH21 1HD,Stone Lane Ind. Estate,Wimborne
EX16 6PQ,Ham Place,Tiverton
```

Promotion table.

```
LOAD DATA
INFILE *
APPEND
INTO TABLE PROMOTIONHISTORY
FIELDS TERMINATED BY "," OPTIONALLY ENCLOSED BY '"'
(EMP_ID, PROM_DATE DATE "DD-MM-YYYY", POSITION, DEPARTMENT, MOD, NARRATIVE)
BEGINDATA
T1,02-02-2001,Junior Developer,Java,0,no comment
T2,13-01-2001,Janitor,Admin,0,no comment
T2,01-02-2001,Manager,Legacy,0,amazing skills
T6,01-02-2001,Janitor,Admin,0,lack of any useful skills
T1,06-03-2001,Developer,Java,0,no comment
T3,03-03-2001,Junior Developer,Java,0,no comment
T3,05-03-2001,Junior Developer,Legacy,0,change of position
T3,06-04-2001,Developer,Legacy,0,avans
T4,18-04-2001,Developer,Legacy,0,no comment
T3,19-04-2001,PA,Admin,0,demoted
T5,20-04-2001,Developer,C#,0,no comment
T5,19-07-2001,Senior Developer,C#,0,promotion
```

Reason table.

```
LOAD DATA
INFILE *
APPEND
INTO TABLE REASON
FIELDS TERMINATED BY "," OPTIONALLY ENCLOSED BY '"'
(CODE, REASON, PAID)
BEGINDATA
A1,Minor illness,Y
A2,Major illness,Y
A3,Death in the family,Y
B1,Thirsday 'illness',N
B2,Friday 'illness',N
```

Refindex table.

```
LOAD DATA
INFILE *
APPEND
INTO TABLE REFINDEX
FIELDS TERMINATED BY "," OPTIONALLY ENCLOSED BY '"'
(EMP_ID, ID, NAME)
BEGINDATA
T1,AA01,Mr Ian McKenzie
T1,AA02,Doctor Evil
T1,AA03,Oxford Brookes
T3,AA04,Mr Mike Brown
T3,AA05,Miss Foo
T4,0021,Mr Bar
T5,AA07,Julian Cook
T6,AA08,Carrie Stealey
T6,AA09,IBM Research
T4,AA10,Bank of India
```

Refloc table.

```
LOAD DATA
INFILE *
APPEND
INTO TABLE REFLOC
FIELDS TERMINATED BY "," OPTIONALLY ENCLOSED BY '"'
(ID, LOCATION)
BEGINDATA
AA01,1344
AA02,8875
AA03,9910
AA04,1231
AA05,8373
AA06,0021
AA07,B003
AA08,6532
AA09,9103
AA10,B220
```

Salary table.

```
LOAD DATA
INFILE *
APPEND
INTO TABLE SALARY
FIELDS TERMINATED BY "," OPTIONALLY ENCLOSED BY '"'
(POSITION, DEPARTMENT, SALARY)
BEGINDATA
Janitor,Admin,12000
Secretary,Admin,18000
PA,Admin,20000
Junior Developer,Java,19000
Junior Developer,Legacy,18000
Developer,Java,21000
Developer,Legacy,24000
Developer,C#,22000
Senior Developer,Java,35000
Senior Developer,Legacy,40000
Senior Developer,C#,37500
Manager,Legacy,40000
Manager,Java,35000
Manager,C#,42000
```

Training table.

```
LOAD DATA
INFILE *
APPEND
INTO TABLE TRAININGHISTORY
FIELDS TERMINATED BY "," OPTIONALLY ENCLOSED BY '"'
(EMP_ID, C_START DATE "DD-MM-YYYY", C_NAME, C_END DATE "DD-MM-YYYY",
LOCATION, RESULT)
BEGINDATA
T1,10-02-2001,Basic of C#,20-02-2001,Banbury,Pass
T1,12-02-2001,Topics in C#,22-02-2001,Banbury,Pass
T1,10-07-2001,Health and Safty in IT,12-07-2001,Office,Pass
T2,10-07-2001,Health and Safty in IT,12-07-2001,Office,Pass
T3,10-07-2001,Health and Safty in IT,12-07-2001,Office,Pass
T4,10-07-2001,Health and Safty in IT,12-07-2001,Office,Pass
T5,10-07-2001,Health and Safty in IT,12-07-2001,Office,Pass
T3,11-05-2001,Adv. topics in .NET,01-06-2001,Bicester,Pass
```