

ជំពូកទី ៨

ការចង្អុល (Data Design)

ក្រោយបញ្ចប់មេរៀននេះ អ្នកនឹងសិក្សា ៖

- Data Design Concepts
- Data Design Terminology
- Data Relationships Normalisation
- Steps in Database Design

នៅក្នុង systems analysis phase យើងធ្វើការបង្កើត data flow diagrams ព្រមទាំងកំណត់ data elements, data flows និង data stores ដើម្បីបង្កើត logical design សំរាប់ប្រព័ន្ធព័ត៌មាន ហើយនៅក្នុងផ្នែកនេះនៃ systems design phase យើងនឹងបង្កើត physical plan for data organization, storage, and retrieval ។ ជំពូកនេះចាប់ផ្តើម ជាមួយនឹងការរំលឹកឡើងវិញពី data design concepts and terminology បន្ទាប់មកពិភាក្សា អំពីទំនាក់ទំនងរវាង data objects ទាំងនោះ ព្រមទាំងវិធីដើម្បីគូរ entity-relationship diagrams ។ យើងនឹងសិក្សាពីវិធីប្រើប្រាស់ normalization concepts ដើម្បីបង្កើត effective database design ។

1. Data Design Concepts

មុនពេលចាប់ផ្តើមបង្កើតប្រព័ន្ធព័ត៌មាន (construct an information system) អ្នកវិភាគប្រព័ន្ធត្រូវតែយល់អោយបានច្បាស់នូវ basic data design concepts រួមមាន៖ តើទិន្នន័យត្រូវរៀបចំជាមធ្យមសម្ព័ន្ធដោយរបៀបណា ព្រមទាំងលក្ខណៈនៃ file-oriented and database systems (how data is structured and the characteristics of file-oriented and database systems) ។

1. 1. Data Structures

File ផ្ទុកទិន្នន័យអំពីមនុស្ស, ទីកន្លែង, វត្ថុ និងព្រឹត្តិការណ៍ដែលមានទំនាក់ទំនងជាមួយ ប្រព័ន្ធព័ត៌មាន។ ឧទាហរណ៍៖ file ប្រហែលជាផ្ទុកទិន្នន័យអំពី Customers, Products, Orders, or Suppliers ជាដើម។ ប្រព័ន្ធព័ត៌មានអាចកសាងក្នុងទម្រង់ជា file-oriented system ឬ database system អាស្រ័យទៅលើ business requirements ប៉ុន្តែប្រព័ន្ធភាគច្រើនគេកសាង វាជាទម្រង់ database system ។

File-oriented system ដំណើរការ data files រៀងៗខ្លួនមួយៗច្រើន ដោយប្រើប្រាស់ វិធីមួយហៅថា **file processing**។ នៅក្នុងឧទាហរណ៍បង្ហាញក្នុងរូប 8.1 យើងឃើញថា repair shop ប្រើប្រាស់ file-oriented systems 2 ដាច់ដោយឡែកពីគ្នាគឺ Work Records system សំរាប់រក្សា JOB data file និង Employee Records system សំរាប់រក្សា MECHANIC data file ។ គួរកត់សំគាល់ថា ព័ត៌មានមួយចំនួនស្ទើរតែផ្ទុកនៅក្នុង data files ទាំងពីរ។

JOB
 Job No
 Work Code
 Hours
 Date
 Mechanic No
 Name
 Pay Rate

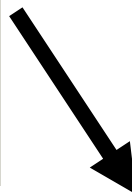
File-oriented Work Records System



Job No	Work Code	Hours	Date	Mechanic No	Name	Pay Rate
134	ALIGN	3.0	12/05/2004	23	Smith Stacy	\$15.50
110	BRAKES	4.0	12/05/2004	17	Jones Jim	\$15.00
198	TUNE	3.2	12/06/2004	12	Lear Robert	\$14.00

MECHANIC
 Mechanic No
 Name
 Pay Rate
 Hire Date
 Status
 Insurance

File-oriented Employee Records System

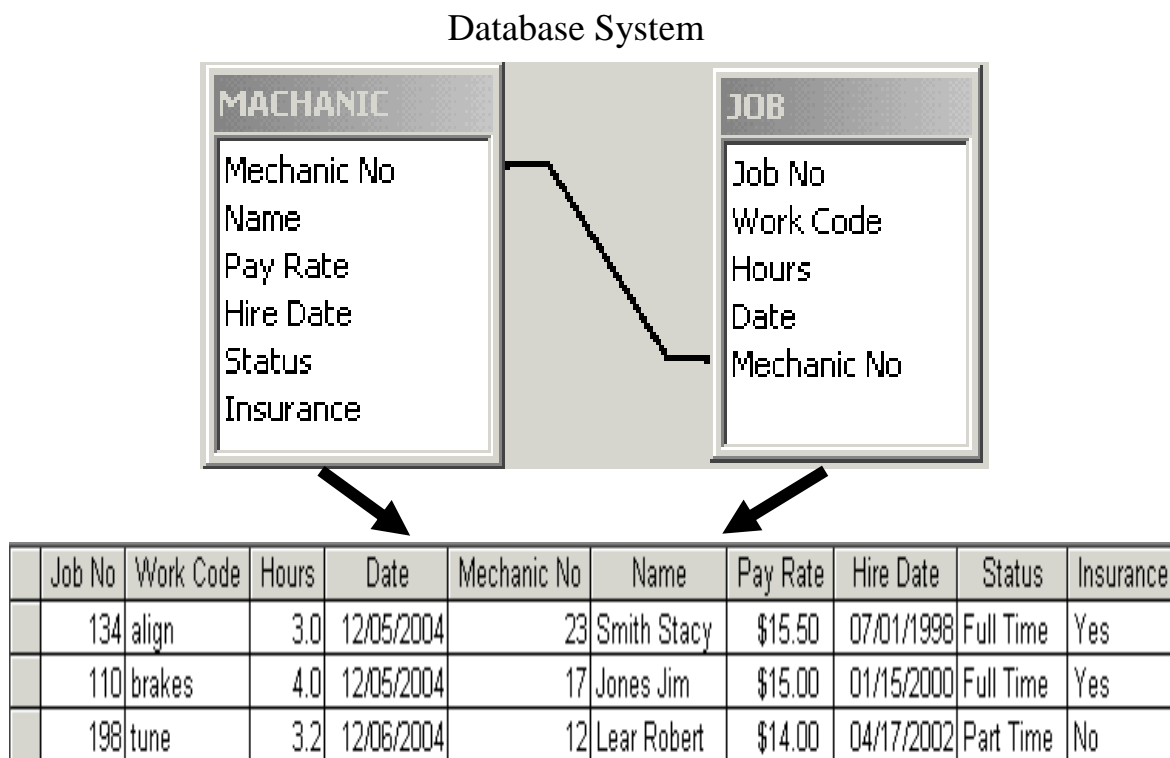


Mechanic No	Name	Pay Rate	Hire Date	Status	Insurance
12	Lear Robert	\$14.00	04/17/2002	Part Time	No
17	Jones Jim	\$15.00	01/15/2000	Full Time	Yes
23	Smith Stacy	\$15.50	07/01/1998	Full Time	Yes

(រូប 8.1)

Database ផ្ទុកនូវ data files តភ្ជាប់គ្នា គេអាចហៅផងដែរថា **tables** ដែលប្រើសំរាប់បង្កើត overall data structure ។ Database environment ផ្តល់នូវ great flexibility and efficiency ប្រសិនបើប្រៀបធៀបជាមួយ file processing ។ ឧទាហរណ៍: រូប 8.2 បង្ហាញពី database ដែលតភ្ជាប់រវាង MECHANIC file និង JOB file ។ គួរកត់សំគាល់ថា database design ជៀសវាងទិន្នន័យស្ទួន (duplication) ដូច្នេះ MECHANIC and JOB files ក្លាយទៅជាសមាសភាគនៃ larger data structure ។ **Database management system**

(DBMS) គឺជាបណ្តុំ tools, features និង interfaces ដែលធ្វើអោយអ្នកប្រើប្រាស់អាច មានលទ្ធភាពបញ្ចូល, កែប្រែ, គ្រប់គ្រង, ប្រើប្រាស់ និងវិភាគទិន្នន័យនៅក្នុង database ។ គេនៅតែប្រើ file processing system ដើម្បីធ្វើការជាមួយ specific applications ប៉ុន្តែសព្វថ្ងៃនេះ ប្រព័ន្ធត្រឹមត្រូវមានភាគច្រើនកសាងទំរង់ជា database ។



(រូប 8.2)

1. 2. Overview of File Processing

ប្រព័ន្ធមួយចំនួនប្រើប្រាស់ file processing ដើម្បីដោះស្រាយ large volumes of structured data លើមូលដ្ឋានទៀងទាត់។ ប្រព័ន្ធចាស់ៗភាគច្រើនប្រើប្រាស់ file processing designs ពីព្រោះវិធីនេះវាសមជាមួយ mainframe hardware និង batch input ។ ទោះបីជា សព្វថ្ងៃនេះគេមិនសូវប្រើប្រាស់វាក៏ដោយ ក៏ file processing ផ្តល់ផលល្អជាង (more efficient) និងមានតំលៃថោកជាង DBMS ក្នុងស្ថានភាពការណ៍មួយចំនួន។

នៅក្នុង file processing environment ក្រុមហ៊ុនជាទូទៅប្រហែលជាមានច្រើន departments ហើយ department នីមួយៗសុទ្ធតែប្រើប្រាស់ប្រព័ន្ធត្រឹមត្រូវមាន និង data files រៀងៗខ្លួន ដូច្នេះធ្វើអោយកើតមានបញ្ហាមួយចំនួនចំពោះ file-processing environment ។

- បញ្ហាទីមួយគឺ **data redundancy** មានន័យថា ទិន្នន័យមួយចំនួនដែលប្រើប្រាស់ ទៅលើប្រព័ន្ធត្រឹមត្រូវមានពីរ វិច្ឆ័យច្រើនត្រូវតែរក្សាទុកច្រើនកន្លែង។ Data redundancy ទាមទារ

storage space បន្ថែម ព្រមទាំងការកែប្រែ និងថែរក្សា (update and maintain) ទិន្នន័យនៅច្រើនកន្លែងមានតំលៃថ្លៃ។

- បញ្ហាទីពីរ **data integrity** អាចកើតឡើង ប្រសិនបើការកែប្រែមិនបានធ្វើទៅគ្រប់ data files ដែលពាក់ព័ន្ធទាំងនោះ។ ការកែប្រែទិន្នន័យចំពោះមួយកន្លែងនឹងបង្កអោយមាន inconsistent data និងផ្តល់លទ្ធផលព័ត៌មានមិនត្រឹមត្រូវនៅក្នុងប្រព័ន្ធផ្សេងទៀត។

- បញ្ហាទីបី **rigid data structure** កើតមានចំពោះ file-processing environment ។ អាជីវកម្មត្រូវតែធ្វើការសំរេចចិត្តដោយសំអាងទៅលើទិន្នន័យប្រើប្រាស់ពាសពេញក្រុមហ៊ុន (company-wide data) និង managers ជារឿយៗត្រូវការព័ត៌មានពី multiple business units and departments ដូច្នេះប្រសិនបើចង់បានព័ត៌មានបែបនោះ ធ្វើអោយ file-based systems ដំណើរការយឺត និងផ្តល់ផលមិនល្អ (inefficient) ដោយសារតែវាច្រើនទទួលព័ត៌មានចេញតែពីមួយផ្នែក។

1. 3. Overview of Database Systems

Database ដែលបានកសាងត្រឹមត្រូវ នឹងផ្តល់នូវដំណោះស្រាយចំពោះបញ្ហាកើតមានលើ file processing ទោះបីជាយ៉ាងណាក៏ដោយ យើងគួរចងចាំនៅក្នុងចិត្តថា ប្រសិនបើយើងកសាង database មិនបានត្រឹមត្រូវតាមក្បួនខ្នាត នឹងបង្កអោយមានបញ្ហាជាច្រើនដូចទៅនឹង file-based system ដែរ។ Database ផ្តល់ overall framework ដែលជៀសវាង data redundancy និងទ្រទ្រង់ real-time environment ។

Data files ត្រូវបានគេកសាងដើម្បីបំពេញចំពោះ individual business systems ចំពោះ file-processing environment ប៉ុន្តែចំពោះ database environment ប្រព័ន្ធជាច្រើនត្រូវបានគេកសាងបញ្ចូលទៅជាប្រព័ន្ធតែមួយ។

ទស្សនៈរបស់អ្នកប្រើប្រាស់តែងតែគិតថា គុណសម្បត្តិជាចំបង (main advantages) នៃ DBMS គឺផ្តល់នូវការប្រើប្រាស់ទិន្នន័យទាន់លឿនពេលវេលា, ទាក់ទងគ្នាទៅវិញទៅមក និងអាចប្រែប្រួលបាន (timely, interactive, and flexible data access)។ DBMS advantages មួយចំនួនមានដូចជា:

- *Control of data redundancy*

ការប្រើប្រាស់ file-oriented system មានការខាតបង់ space ជាច្រើនដោយសារការរក្សាទុកព័ត៌មានដូចគ្នាក្នុង files លើសពី 1 កន្លែង។ ផ្ទុយទៅវិញ database approach មានគោលបំណងលុបបំបាត់ចោលភាពស្ងួតទិន្នន័យ (eliminate the redundancy) ដោយច្របាច់

files បញ្ចូលគ្នាធ្វើអោយវាមិនបានរក្សាទុក ច្បាប់ចំលងទិន្នន័យដូចគ្នាជាច្រើន (several copies of the same data)។ ទោះបីជាយ៉ាងណាក៏ដោយ database approach មិនបានលុបបំបាត់ចោលភាពស្មុន់ស្រឡាយទាំងស្រុងនោះទេ ប៉ុន្តែវាអាចគ្រប់គ្រងលើចំនួនភាពស្មុន់ស្រឡាយក្នុង database បាន ប៉ុន្តែជូនកាលវាមានភាពចាំបាច់ក្នុងការផ្ទុក data items ដើម្បី improve performance ។

- *Data consistency*

នៅពេលដែលយើងអាចលុបភាពស្មុន់ស្រឡាយ គ្រប់គ្រងភាពស្មុន់ស្រឡាយ (eliminating or controlling redundancy) ពេលនោះយើងបានកាត់បន្ថយ risk of inconsistencies occurring ។ *ប្រសិនបើយើងកែប្រែទិន្នន័យ នោះយើងកែប្រែទិន្នន័យតែ 1 កន្លែងទេ* ប៉ុន្តែផ្ទុយទៅវិញប្រសិនបើ data items រក្សាទុកលើសពី 1 កន្លែង ហើយប្រព័ន្ធមិនបានដឹងពីបញ្ហានេះ នោះប្រព័ន្ធត្រូវតែប្រាកដថាវាលំប្រាប់ចំលង item (all copies of the item) រក្សា consistent ។ គួរអោយសោកស្តាយព្រោះ DBMS ភាគច្រើនមិនបានរក្សា consistency ដោយស្វ័យប្រវត្តិទេ។

- *More information from the same amount of data*

នៅពេលយើងច្របាច់ (ផ្គុំ) operational data បញ្ចូលគ្នាធ្វើអោយ organization មានលទ្ធភាពក្នុងការបង្កើតព័ត៌មានបន្ថែម (derived additional information) ចេញពីទិន្នន័យដូចគ្នា។

- *Sharing data*

ជាទូទៅ people រឺ department ដែលប្រើ files ច្រើនតែជាម្ចាស់កម្មសិទ្ធិ files នោះផ្ទុយទៅវិញ database ដែលជាកម្មសិទ្ធិនៃ organization ទាំងមូលត្រូវតែអាចប្រើប្រាស់ដោយ authorized users ។ ប្រសិនបើមាន users ច្រើន ពេលនោះការប្រើប្រាស់ទិន្នន័យក៏កាន់តែច្រើនដែរ ហើយយើងអាចបង្កើត application ថ្មីប្រើប្រាស់ទិន្នន័យក្នុង database មានស្រាប់ និង បន្ថែមទិន្នន័យដែលមិនទាន់បានរក្សាទុកជាជាងធ្វើការកំណត់រកទិន្នន័យទាំងអស់សារជាថ្មី។

- *Improved data integrity*

Database integrity សំដៅទៅលើ validity និង consistency នៃទិន្នន័យរក្សាទុក។ Integrity ជាទូទៅបង្ហាញតាមរយះពាក្យ constraints (**Constraints** ជា consistent rules កំណត់លើ database មិនអោយបំពាន)។ Constraint អាចកំណត់លើ data items ក្នុង single record រឺ relationship រវាង records ។ ឧទាហរណ៍: integrity constraint កំណត់ថា

employee's salary មិនអាចលើសពី \$4,000 ហើយ DBMS ច្រើនតែធ្វើការ enforce integrity constraints ។

- *Improved security*

Database security សំដៅលើការរារាំងចូលប្រើប្រាស់ទិន្នន័យក្នុង database ពី unauthorized users ។ ប្រសិនបើមិនបានចាត់វិធានការការពារអោយបានសមរម្យនោះទេ ការច្របាច់ទិន្នន័យចូលគ្នាមានលក្ខណៈស្មុគស្មាញជាង file-oriented system ហើយជាទូទៅ DBMS ធ្វើការកំណត់ database security ក្នុងទម្រង់ជា user names និង passwords សំរាប់កំណត់ authorized user ។ ជាងនេះទៅទៀត យើងអាចកំណត់សិទ្ធិបន្ថែមទៀតទៅលើ authorized user ក្នុងការប្រើប្រាស់ទិន្នន័យដូចជា retrieval, insert, update, delete ។
ឧទាហរណ៍៖ DBA អាចចូលប្រើប្រាស់រាល់ទិន្នន័យទាំងអស់ក្នុង database ប៉ុន្តែ user អាចចូលប្រើប្រាស់ទិន្នន័យដែលពាក់ព័ន្ធនឹងការងាររបស់គាត់តែប៉ុណ្ណោះ។

- *Economy of scale*

ការបញ្ចូលគ្នារាល់ operational data ទាំងអស់នៃ organization ទៅកាន់ database ហើយបង្កើត applications ធ្វើការជាមួយប្រភពទិន្នន័យនោះ អាចជួយកាត់បន្ថយតំលៃក្នុងការចំណាយ។ លុយធ្លាប់ចំណាយទៅលើ department នីមួយៗចំពោះការអភិវឌ្ឍន៍ និងថែរក្សា file-oriented system អាចបញ្ចូលគ្នា ប្រហែលជាទទួលបាន lower total cost នាំអោយមាន economy of scale ។ Combined budget អាចប្រើប្រាស់សំរាប់ទិញ system configuration ដែលសមរម្យទៅនឹងតំរូវការ organization ហើយប្រហែលជាផ្ទុកនូវ one large, powerful computer រឺ a network of smaller computers ។

- *Balance of conflicting requirements*

តំរូវការរបស់ user រឺ department អាចមាន conflict ជាមួយតំរូវការរបស់ user ផ្សេងទៀត។ នៅពេលដែល database ស្ថិតក្រោមការគ្រប់គ្រងរបស់ DBA, DBA អាចសំរេចចិត្តធ្វើការកសាង និងប្រើប្រាស់ database ដែលផ្តល់នូវការប្រើប្រាស់ប្រភពទិន្នន័យល្អបំផុតចំពោះ organization ទាំងមូល។

- *Improved data accessibility and responsiveness*

DBMS ភាគច្រើនផ្តល់នូវ query language រឺ report ដែលអនុញ្ញាតិអោយ users ធ្វើការសាកសួរ ad hoc question (សំនួរដែលកើតឡើងភ្លាមៗ) និងទទួលបានព័ត៌មានទាំងនោះស្ទើរតែភ្លាមៗលើ terminal របស់ពួកគេដោយមិនចាំបាច់ទាមទារអោយ programmers សរសេរ software ទាញព័ត៌មានទាំងនោះពី database ទេ។ ឧទាហរណ៍៖ branch manager អាច

បង្ហាញរាល់ flats ទាំងអស់ដែលមាន monthly rent លើសពី \$400 ដោយឃ្លាបញ្ជា SQL លើ terminal ដូចខាងក្រោម

```
SELECT *  
FROM Property_for_Rent  
WHERE type = 'Flat' AND rent > 400  
- Increased productivity
```

DBMS បានផ្តល់នូវ standard functions មួយចំនួនដែល programmers ជាធម្មតា តែងតែសរសេរនៅក្នុង file-oriented application ។ ការផ្តល់ functions ទាំងនេះនៃ DBMS អនុញ្ញាតិអោយ programmers ជញ្ជីងគិតបន្ថែមទៅលើ specific functions ណាដែល users ត្រូវការដោយគ្មានការអំពី low-level implementation details ។ DBMSs ជាច្រើនបានផ្តល់ ផងដែរនូវ *fourth-generation environment* ដោយផ្ទុកនូវ tools សំរាប់ជួយសំរួល ការអភិវឌ្ឍន៍ database applications ហើយលទ្ធផលនេះធ្វើអោយកើន programmer productivity និងកាត់បន្ថយពេលវេលាកសាង (ជាមួយតំលៃចំណាយផងដែរ) ។

- Improved maintenance through data independence

នៅក្នុង file-oriented system ការពណ៌នាទិន្នន័យ (description of data) និង logic សំរាប់ប្រើប្រាស់ (accessing) ទិន្នន័យត្រូវបានច្របាច់បញ្ចូលទៅជាមួយ application program នីមួយៗ ធ្វើអោយ programs ពឹងពាក់លើទិន្នន័យ។ ប្រសិនបើមានការកែប្រែទៅលើ រចនាសម្ព័ន្ធទិន្នន័យ (structure of the data) ឧទាហរណ៍: កែប្រែ address ដែលកំនត់ 40 តួអក្សរទៅជា 41 តួអក្សរ វិការកែប្រែទៅលើវិធីដែលទិន្នន័យរក្សាទុកលើ disk (a change to the way the data is stored on disk) ត្រូវអោយមានការកែប្រែសមរម្យទៅលើ programs ដែលប៉ះពាល់។ ផ្ទុយទៅវិញចំពោះ DBMS ដែលផ្ទុក data descriptions ដាច់ដោយឡែកពី applications ធ្វើ អោយការកែប្រែ applications មិនប៉ះពាល់ដល់ data descriptions ទេ នេះគឺជាលក្ខណៈ *data independence* ។

- Increased concurrency

នៅក្នុង file-oriented system ប្រសិនបើអនុញ្ញាតិអោយ users 2 រឺច្រើនប្រើប្រាស់ file តែមួយក្នុងពេលដំណាលគ្នានោះ វាអាចមានការប៉ះទង្គិចគ្នាទៅវិញទៅមកបណ្តាលអោយមាន ការបាត់បង់ព័ត៌មាន (loss of information) រឺជាងនេះទៅទៀត loss of integrity ។ DBMS ជាច្រើនអាចគ្រប់គ្រងការចូលប្រើប្រាស់ database ក្នុងពេលតែមួយបាន ហើយអាចការពារបញ្ហា នេះមិនអោយកើតឡើង។

- Improved backup and recovery services

File-oriented systems ជាច្រើនបានដាក់ការទទួលខុសត្រូវទៅលើ users ដើម្បីចាត់វិធានការការពារទិន្នន័យពីបញ្ហាកើតមានឡើងនៅលើ computer system រឺ application program ដែលទាមទារអោយធ្វើការ backup ទិន្នន័យរៀងរាល់យប់។ នៅពេលមានព្រឹត្តិការណ៍បញ្ហាមួយកើតឡើងនៅថ្ងៃបន្ទាប់ នោះយើងត្រូវតែ restore backup ហើយរាល់ការងារដែលបានបញ្ចូលចាប់តាំងពីពេល backup ត្រូវចាត់បង់ដោយទាមទារអោយបញ្ចូលសារជាថ្មី។ ផ្ទុយទៅវិញ modern DBMSs ផ្តល់ facilities ដើម្បីកាត់បន្ថយចំនួនប្រតិបត្តការដែលបានចាត់បង់។

2. Data Design Terminology

អ្នកវិភាគប្រព័ន្ធត្រូវតែជ្រើសរើសវិធីសាស្ត្រមួយដើម្បីចាប់ផ្តើមបង្កើត data management system ដោយប្រើប្រាស់ data design concepts ដែលបានរៀបរាប់ពីមុន ហើយជំហានដំបូងគឺត្រូវស្វែងយល់ពី data design terminology ជាមុនសិន។

1. 1. Definitions

រូបខាងក្រោមបង្ហាញពី table មួយមាន fields និង records ជាច្រើនដែលរៀបរាប់នៅក្នុង entity មួយហៅថា CUSTOMER ។

Entity: CUSTOMER

primary key

fields

	CustomerID	Customer Name	Sex	Phone	Address
+	1	Chay Sarath	Male	(012) 897-232	Happy Supermarket
+	2	Khun Ngeth	Male	(011) 989-897	No. 133, St 123, Phsa They Market
+	3	Poum Sotheary	Female	(012) 893-434	No. 81, St 235, Phsa They Market
+	4	Ros Dany	Female	(012) 786-666	No.18, St 188, Olympic Market
+	5	Mean Sona	Female	(012) 785-555	No.21, St 188, Olympic Market
+	6	Nuan Seang	Female	(011) 998-777	No.85, St 158, Olympic Market

(រូប 8.3)

Entity

Entity សំដៅទៅលើមនុស្ស, ទីកន្លែង, វត្ថុ រឺព្រឹត្តិការណ៍សំរាប់ប្រមូល និងរក្សាទុកទិន្នន័យឧទាហរណ៍: Sales system ផ្ទុក entities មានឈ្មោះថា CUSTOMER, PRODUCT, ORDER, and SUPPLIER។ នៅពេលយើងរៀបចំបង្កើត DFDs ក៏ឡុងពេល systems

analysis phase យើងបានធ្វើការកំណត់ entities and data stores ផ្សេងៗរួចហើយ ក្នុងពេលនេះយើងនឹងពិចារណាទៅលើទំនាក់ទំនង (relationships) រវាង entities ទាំងនោះ។

Field

Field អាចហៅថា **Attribute** ដែលគឺជាចរិតលក្ខណៈ វិហេតុការណ៍តែមួយ (single characteristic or fact) នៃ entity។ នៅក្នុងឧទាហរណ៍ខាងលើ យើងឃើញថា entity CUTOMER មាន fields ជាច្រើនប្រើសំរាប់រក្សាទុកទិន្នន័យចំពោះ customer នីមួយៗ ដូចជា: CustomerName, Sex, and Phone។ **Common field** គឺជា attribute ដែលស្ថិតក្នុង entity លើសពីមួយ ហើយជាទូទៅ common field ត្រូវបានគេប្រើប្រាស់ដើម្បីភ្ជាប់ entities ។

Record

Record អាចហៅថា **Tuple** គឺជាសំនុំនៃ fields មានទំនាក់ទំនងគ្នាប្រើសំរាប់រៀបរាប់ពី one instance វិសមាជិក entity ដូចជា one customer, one order, or one product។ Record មួយអាចមាន field មួយ ឬច្រើនអាស្រ័យទៅតាមតម្រូវការព័ត៌មានចង់បាន។

File and Table

Records ជាច្រើនប្រមូលផ្តុំគ្នាបង្កើតទៅជា table or file ។

នៅក្នុង file-oriented system សំនុំ records ទាក់ទងគ្នាជាច្រើនផ្តុំចូលគ្នាបង្កើតជា file សំរាប់ផ្ទុកទិន្នន័យចំពោះមនុស្ស, ទឹកថ្លៃ, វត្ថុ វិព្រឹត្តិការណ៍ដូចទៅនឹងអ្វីដែលយើងបានសិក្សាកាលពីជំពូកមុន។ ឧទាហរណ៍: ប្រសិនបើ inventory system គ្រប់គ្រង 1500 products ដូច្នេះគេនិយាយថា PRODUCT file មាន 1500 records ដោយមួយ record សំរាប់ទំនិញនីមួយៗ។ ជារឿយៗ ទិន្នន័យនៅ file អាចត្រូវបានគេរៀបចំអោយមានរចនាសម្ព័ន្ធក្នុងទម្រង់មួយដើម្បីបង្កើនល្បឿនដំណើរការនៅក្នុង file-oriented system ។

នៅក្នុង database environment សំនុំ records ទាក់ទងគ្នាជាច្រើនផ្តុំចូលគ្នាបង្កើតជា table សំរាប់រក្សាទុកទិន្នន័យចំពោះ entity ជាក់លាក់ណាមួយ។ ជាទូទៅ table បង្ហាញក្នុងទម្រង់ជា two-dimensional structures ដែលផ្ទុកនូវ vertical columns បង្ហាញពី fields និង horizontal rows បង្ហាញពី records។ ឧទាហរណ៍: sale database system ប្រហែលជាមាន tables ជាច្រើនដូចជា: CUSTOMER, ORDER, PRODUCT, and SUPPLIER ។

2. 2. Key Fields

កំឡុងពេល systems design phase យើងប្រើប្រាស់ **key field** ដើម្បីរៀបចំ, ប្រើប្រាស់ (access) និងថែរក្សា data structure។ Keys ចែកចេញជា 3 ប្រភេទគឺ primary keys, candidate keys, and foreign keys ។

Primary Keys

Primary key គឺជា field មួយ រឺបណ្តុំ fields ដែលធ្វើការកំណត់អត្តសញ្ញាណតែឯកឯង និងតូចបំផុតចំពោះសមាជិក entity។ ឧទាហរណ៍៖ នៅក្នុង Customer table មាន field Customer number គឺជា primary key ពីព្រោះគ្មាន customers ណាដែលមាន customer number ដូចគ្នាទេ។ Primary key ត្រូវតែផ្ទុកព័ត៌មានតិចបំផុត ដោយជ្រើសរើស fields ណាដែលគេត្រូវដើម្បីកំណត់អត្តសញ្ញាណតែឯកឯង។

Primary key អាចកើតពីការផ្សំចូលគ្នារវាង fields ចាប់ពី 2 ឡើងទៅ។ ឧទាហរណ៍៖ ប្រសិនបើសិស្សចុះឈ្មោះវគ្គសិក្សាចំនួន 3 នោះ student number នឹងបង្ហាញដល់ទៅ 3 records នៅក្នុង registration system។ ប្រសិនបើវគ្គសិក្សាមានសិស្ស 20 នាក់ នោះ 20 separate records នឹងលេចឡើងចំពោះ course number នោះជាមិនខានដោយ record មួយ សំរាប់សិស្សនីមួយៗដែលបានចុះឈ្មោះ។

នៅក្នុង registration file ទាំង student number រឺ course ID សុទ្ធតែមិនអាចកំណត់អត្តសញ្ញាណបានទេ ដូច្នេះវាសិនមួយមិនអាចកំណត់ជា primary key បានទេ។ ដើម្បីកំណត់សិស្សណាមួយនៅក្នុងវគ្គជាក់លាក់ណាមួយ យើងត្រូវតែផ្សំគ្នារវាង student number និង course ID ដើម្បីដើរតួនាទីជា primary key ។ នៅក្នុងករណីនេះ primary key គឺជា **combination key** រឺ **multivalued key** ។

រួមខាងក្រោមបង្ហាញពី tables ចំនួន 4 ខុសគ្នាដោយ tables ចំនួន 3 ដំបូងមាន primary key កើតពី field តែមួយគត់ (single-field primary key) និង table ទី 4 មាន primary key កើតពីការផ្សំគ្នារវាង foreign key fields ចំនួន 2: STUDENT-NUMBER and COURSE-ID ។

STUDENT Table

<u>STUDENT- NUMBER</u>	STUDENT- NAME	TOTAL- CREDITS	ADVISOR- NUMBER
1036	Linda Marie	14	49
3397	Sam Carr	9	49
4070	Kelly Horowitz	8	23

ADVISOR Table

<u>ADVISOR- NUMBER</u>	SOCIAL- SECURITY- NUMBER	ADVISOR- NAME
23	504931227	Alice Jones
49	036771990	Carlton Smith

COURSE Table

<u>COURSE- ID</u>	COURSE DESCRIPTION	NUMBER-OF- CRDITS
CHM112	General Chemistry I	5
CSC151	Computer Science I	3
ENG101	English Composition	3
MKT212	Basic Marketing	3

GRADE Table

STUDENT- NUMBER	COURSE-ID	GRADE
1035	CSC151	B
1035	MKT212	A
1035	ENG101	B
1035	CHM112	A
3397	ENG101	A
3397	MKT212	C
3397	CSC151	B
4070	CSC151	B
4070	CHM112	C

(Figure 8.4)

Candidate Keys

នៅក្នុងករណីមួយចំនួន យើងអាចជ្រើសរើសយក field មួយ រឺបណ្តុំ fields ជាច្រើន ដើម្បីកំណត់ជា primary key។ Fields ទាំងឡាយណាដែលអាចដើរតួនាទីជា primary key គេហៅថា **candidate key**។ ឧទាហរណ៍៖ ប្រសិនបើ employee នីមួយៗមាន unique employee number ដូច្នេះយើងអាចកំណត់ employee number រឺ Social Security number ធ្វើជា primary key។ ដោយសារតែនៅក្នុង table មួយមាន primary key តែមួយគត់ ដូច្នេះយើងត្រូវជ្រើសរើស field ណាដែលផ្ទុកទិន្នន័យតិចបំផុត និងងាយស្រួលប្រើប្រាស់ លើសពីនេះទៀត field ណាមិនត្រូវបានជ្រើសរើសជា primary key គេហៅថា **nonkey field** ។

Foreign Keys

គួរកត់សំគាល់ថា common field លេចឡើងនៅក្នុង tables ច្រើនជាងមួយ និងត្រូវបាន គេប្រើប្រាស់ដើម្បីបង្កើតទំនាក់ទំនង (relationship or link) រវាង tables ទាំងនោះ។ ឧទាហរណ៍៖ នៅក្នុងរូប 8.4 ខាងលើយើងសង្កេតឃើញថា ADVISOR-NUMBER field ស្ថិតនៅក្នុង tables ទាំង 2 គឺ STUDENT table និង ADVISOR table ហើយ field នេះ ប្រើសំរាប់ភ្ជាប់ tables ទាំង 2 នេះ។ គួរកត់សំគាល់ថា ADVISOR-NUMBER field គឺជា primary key នៅក្នុង ADVISOR table ដែលប្រើប្រាស់កំណត់អត្តសញ្ញាណ advisor នីមួយៗ និងវាមានតួនាទីជា foreign key នៅក្នុង STUDENT table។ **Foreign key** គឺជា field ស្ថិតនៅក្នុង table មួយដែលត្រូវតែផ្គូផងជាមួយតំលៃ primary key នៅក្នុង table មួយផ្សេងទៀតដើម្បីបង្កើតទំនាក់ទំនងរវាង tables ទាំង 2 ។

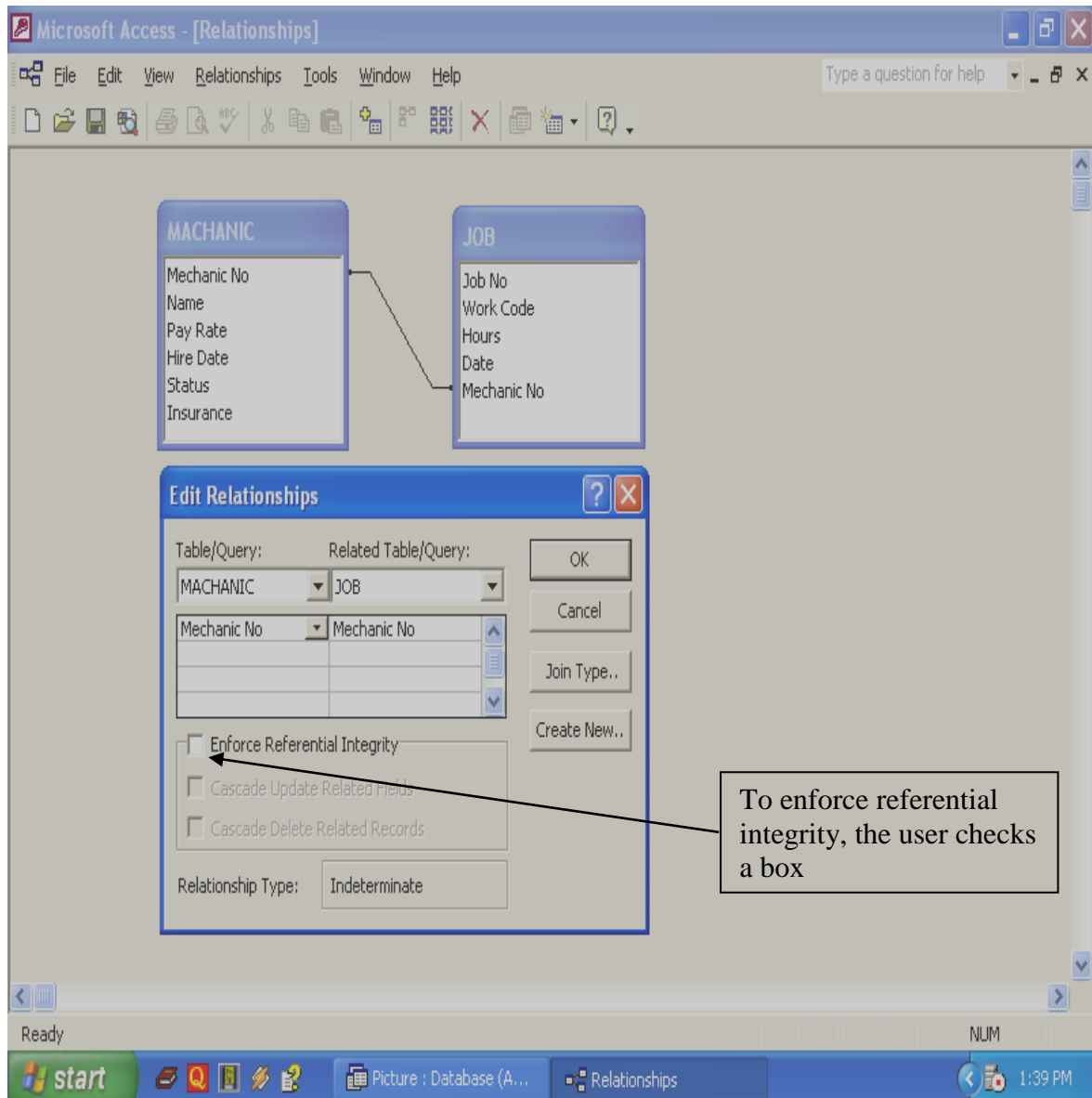
Foreign key មិនចាំបាច់មានឯករាជ្យ (unique) ដូច primary key ទេ។ ឧទាហរណ៍៖ Carlton Smith មាន advisor number 49 ហើយតំលៃ 49 នេះត្រូវមានតែមួយគត់នៅក្នុង ADVISOR table ពីព្រោះវាគឺជា primary key ប៉ុន្តែតំលៃ 49 អាចលេចឡើងច្រើនលើក ច្រើនសារនៅក្នុង STUDENT table ដែលជាទីកន្លែង advisor number ដើរតួនាទីជា foreign key។ រូប 8.4 ខាងលើបង្ហាញពី foreign keys 2: STUDENT-NUMBER field and COURSE-ID field ចូលរួមគ្នាបង្កើត primary key នៅក្នុង GRADE table ដោយ STUDENT-NUMBER field ត្រូវតែផ្គូផងជាមួយ student number នៅក្នុង STUDENT table និង COURSE-ID field ត្រូវតែផ្គូផងជាមួយ course ID នៅក្នុង COURSE table ។

Foreign keys are not unique. Advisor number ជាកំណត់ណាមួយអាចបង្ហាញឡើងច្រើនលើកនៅក្នុង STUDENT table ដោយការបង្ហាញចេញមួយលើកសំដែងពី student នីមួយៗត្រូវបានគេចាត់តាំងទៅជាមួយ advisor។ នៅក្នុង GRADE table មាន foreign keys student number and course ID លេចឡើងច្រើនលើកច្រើនសារ ប៉ុន្តែ foreign keys ទាំង 2 នេះចូលរួមគ្នាបង្កើតជា primary key ដូច្នេះ student ជាកំណត់ណាមួយអាចសិក្សា course ណាមួយបានតែម្តងប៉ុណ្ណោះ។

2. 3. Referential Integrity

កាលពីនៅក្នុងជំពូកទី 7 យើងបានសិក្សារួចហើយចំពោះ validity check ដែលប្រើសំរាប់ការការពារ data input errors ហើយនៅក្នុងមានប្រភេទមួយនៅក្នុង validity check គេហៅថា **referential integrity** ដែលគឺជាសំនុំច្បាប់សំរាប់ជៀសវាង data inconsistency and quality problem។ Referential integrity នៅក្នុង relational database មានន័យថាយើងមិនអាចបញ្ចូលតំលៃ foreign key ទៅក្នុង table បានទេ ប្រសិនបើតំលៃនោះមិនត្រូវគ្នាជាមួយតំលៃ primary key កំពុងមាននៅក្នុង table មួយផ្សេងទៀត។ ឧទាហរណ៍: referential integrity ការពារមិនអោយបញ្ចូល customer order ទៅក្នុង order table ប្រសិនបើ customer នោះមិនទាន់មានព័ត៌មានផ្ទុកនៅក្នុង customer table។ ប្រសិនបើគ្មាន referential integrity ទេ យើងប្រហែលជាផ្ទុក order មានលក្ខណៈ **orphan** ពីព្រោះវាគ្មានបញ្ជាក់ពីព័ត៌មាន customer ពាក់ព័ន្ធដែលធ្វើការកម្លាំងទិញ។

Referential integrity នៅក្នុងឧទាហរណ៍ 8.4 មិនអនុញ្ញាតិអោយអ្នកប្រើប្រាស់បញ្ចូល advisor number (foreign key value) នៅក្នុង STUDENT table ទេប្រសិនបើគ្មាន valid advisor number (primary key value) មាននៅក្នុង ADVISOR table។ Referential integrity ក៏រារាំងមិនអោយលុប record ណាដែលមាន primary key ផ្លូវផ្តងជាមួយ foreign key នៅក្នុង table មួយផ្សេងទៀត។ ឧទាហរណ៍: ឧបមាថា advisor សំរេចចិត្តលាឈប់ទៅធ្វើការនៅសាលាផ្សេងទៀត ដូច្នេះយើងមិនអាចលុប advisor នោះចេញពី ADVISOR table បានទេខណៈដែល STUDENT table នៅតែប្រើប្រាស់ advisor number នោះ។ ដើម្បីជៀសវាងបញ្ហានេះ យើងត្រូវចាត់តាំង student នោះទៅកាន់ advisor ផ្សេងទៀតដោយផ្លាស់ប្តូរតំលៃ ADVISOR-NUMBER field មុននឹងអាចលុប record នោះចោលបាន។



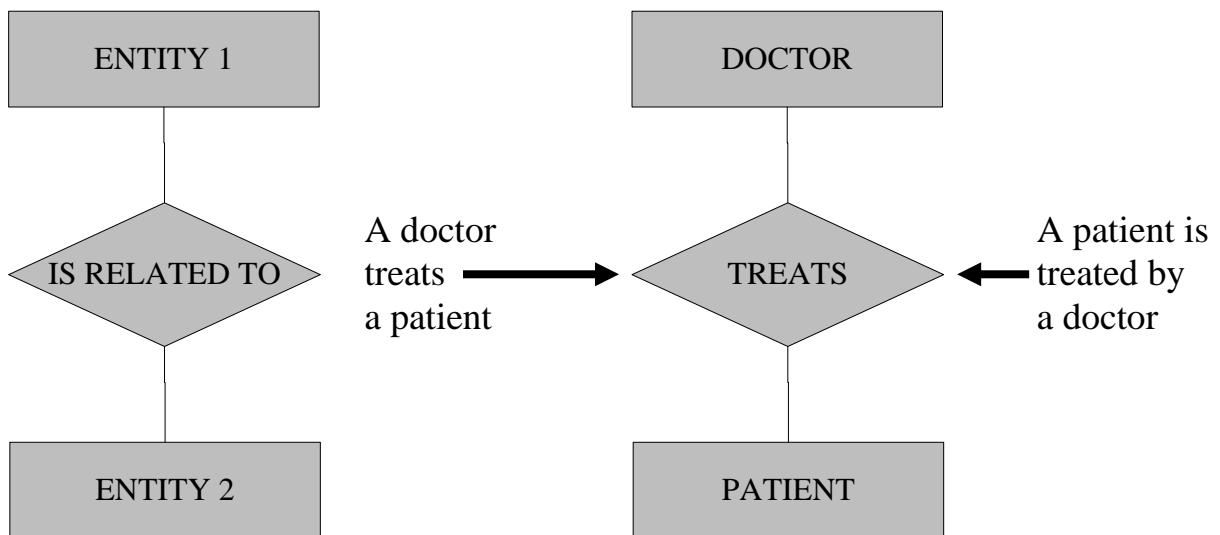
(រូប 8.5)

3. Data Relationships

គួរនឹករលឹកឡើងវិញថា entity សំដៅទៅលើមនុស្ស, ទីកន្លែង, វត្ថុ ឬព្រឹត្តិការណ៍ចំពោះទិន្នន័យដែលយើងបានប្រមូលកំឡុងពេល systems analysis phase នៅក្នុងជំពូកទី 3 បរិយាយពី requirement modeling ។ **Relationship** គឺជាការភ្ជាប់ (logical link) រវាង entities ឧទាហរណ៍: relationship កើតមានរវាង entities PRODUCT and WAREHOUSE ពីព្រោះ products រក្សាទុកនៅក្នុង warehouse ។

3. 1. Entity-Relationship Diagrams

Entity-relationship diagram (ERD) គឺជាការបង្ហាញជាលក្ខណៈរូបភាពចំពោះប្រព័ន្ធព័ត៌មានដោយបង្ហាញពីទំនាក់ទំនងរវាង entities ទាំងនោះ ហើយរូប 8.6 បង្ហាញពីទំរង់ជាមូលដ្ឋានចំពោះ ERD ។ រាល់ entity ទាំងអស់បង្ហាញជារាងចតុកោណកែង (rectangle) ព្រមទាំងឈ្មោះជា singular noun និង diamond បង្ហាញពី relationship ព្រមទាំងឈ្មោះបង្ហាញជា active verb ។ ឧទាហរណ៍: រូប 8.6 បង្ហាញពីវេជ្ជបណ្ឌិតព្យាបាលអ្នកជំងឺ (doctor treats a patient) ។ យើងក៏អាចនិយាយថា អ្នកជំងឺត្រូវបានព្យាបាលដោយវេជ្ជបណ្ឌិត ប៉ុន្តែជារឿយៗគេច្រើនតែប្រើប្រាស់ active verb ពីព្រោះវាមានលក្ខណៈស្តង់ដារ។ Entity-relationship diagram មិនបានពណ៌នាពី data or information flow ទេ ហើយ ERD គ្មានប្រើសញ្ញាក្បាលព្រួញទេ លើសពីនេះទៀត យើងអាចរៀបចំទីតាំង entities នៅខាងលើ រឺខាងច្រើនតាមបំណងរបស់យើងដោយលែបាំងណាអោយងាយមើល។

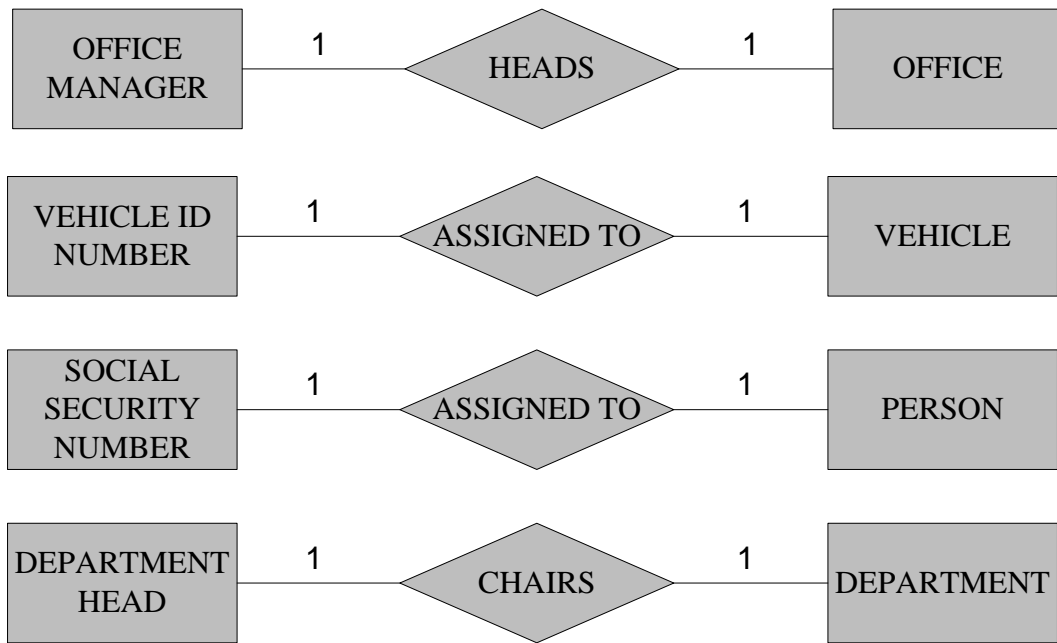


(រូប 8.6)

Relationships ចែកជា 3 ប្រភេទចំបងគឺ:

One-to-one relationship

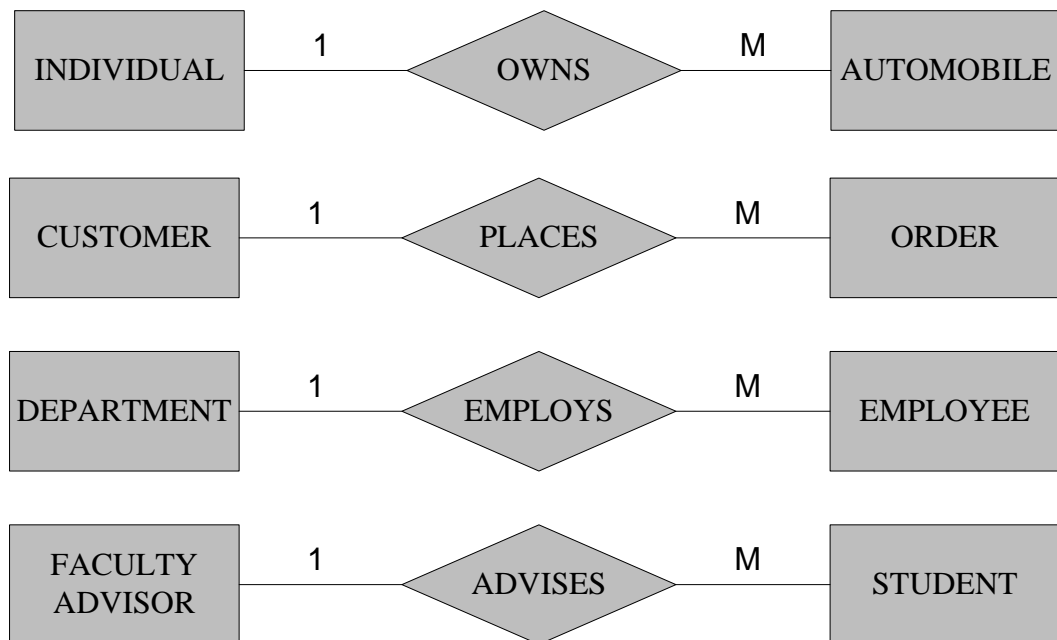
សរសេរអក្សរកាត់ជា 1:1 ដែលកើតឡើងនៅពេល record នីមួយៗនៅក្នុង entity ទីមួយទាក់ទងជា record តែមួយគត់នៅក្នុង entity ទីពីរ ហើយ record នីមួយៗនៅក្នុង entity ទីពីរទាក់ទងជាមួយ record តែមួយគត់នៅក្នុង entity ទីមួយ។ ERDs ចំពោះ 1:1 entity relationships បង្ហាញនៅរូបខាងក្រោម។ តំលៃលេខ 1 គេដាក់ក្បែររូបម្ចាត់ទាំង 2 ដែលតភ្ជាប់ពី rectangle ទៅកាន់ diamond ដើម្បីបញ្ជាក់ 1:1 relationship ។



(រូប 8.7)

One-to-many relationship

សរសេរអក្សរតាត់ជា 1:M ដែលកើតឡើងនៅពេល record មួយនៅក្នុង entity ទីមួយ ទាក់ទងជា record ជាច្រើននៅក្នុង entity ទីពីរ ហើយ record នីមួយៗនៅក្នុង entity ទីពីរ ទាក់ទងជាមួយ record តែមួយគត់នៅក្នុង entity ទីមួយ។



(រូប 8.8)

ឧទាហរណ៍៖ ទំនាក់ទំនងរវាង DEPARTMENT និង EMPLOYEE គឺជា one-to-many relationship មានន័យថា department មួយមាន employees ជាច្រើន ប៉ុន្តែ

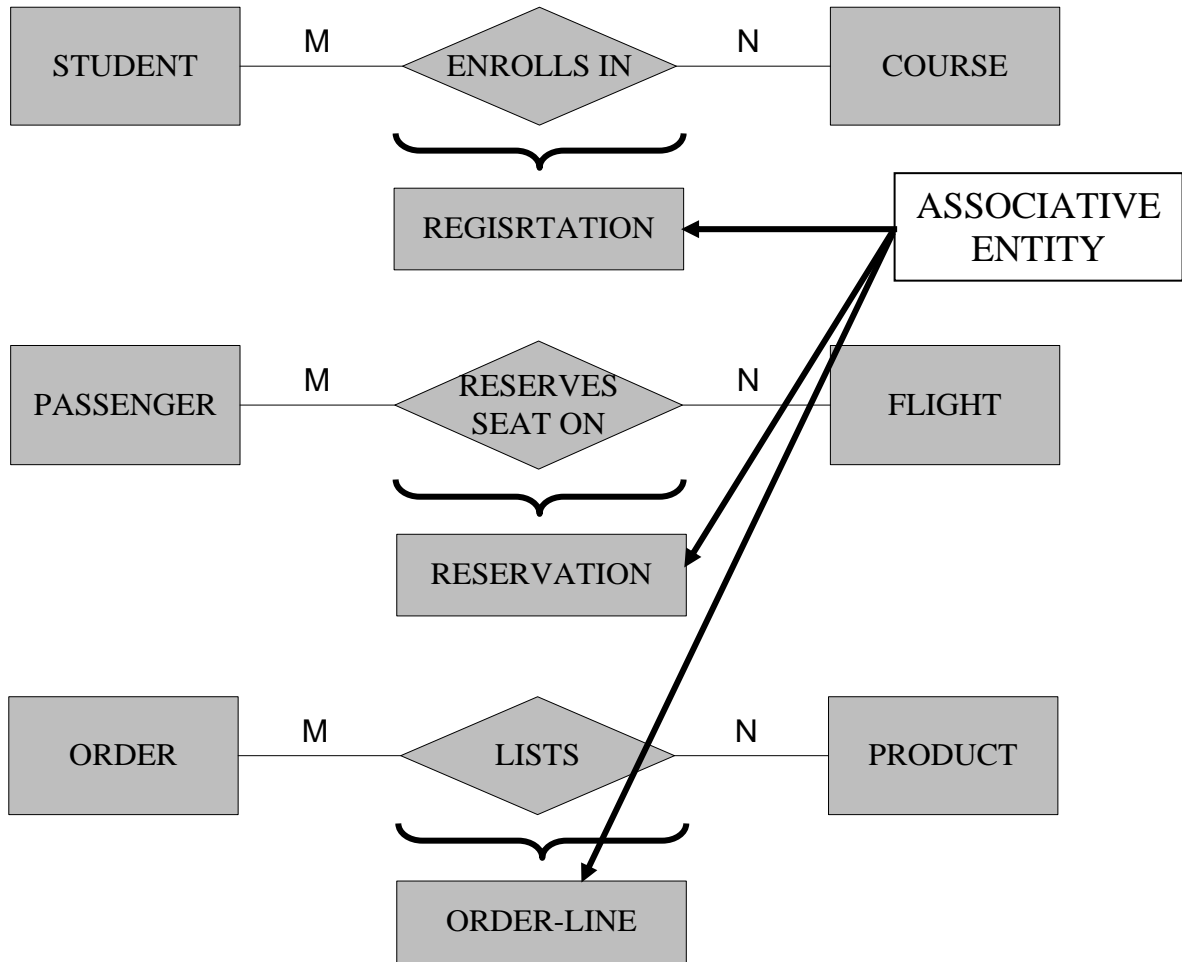
employee នីមួយៗធ្វើការនៅក្នុង department តែមួយគត់។ រូបខាងលើបង្ហាញពីរូបភាព 1:M entity-relationship diagrams ជាច្រើន។ បន្ទាត់តភ្ជាប់ទៅកាន់ many entity គេដាក់តួអក្សរ M និងតំលៃលេខ 1 គេដាក់នៅខាង one entity វិញដើម្បីបញ្ជាក់ 1:M relationship។ មាន software programs មួយចំនួននិមិត្តសញ្ញាតំណាងខាង many side នៅក្នុង relationship គេដាក់តួអក្សរអានន៍ “∞”។

តើមានចំនួនប៉ុន្មានទៅចំពោះ many? ចំពោះរូប 1:M relationship បង្ហាញខាងលើ បញ្ជាក់ថា automobile មួយមានម្ចាស់កម្មសិទ្ធិតែមួយនាក់គត់ ប៉ុន្តែ person ម្នាក់អាចគ្មាន automobile សោះ រឺមានមួយ រឺ 20 ហេតុដូច្នេះ many មានន័យថា ចំនួនណាមួយដោយចាប់ ផ្តើមពីលេខសូន្យ “0”។

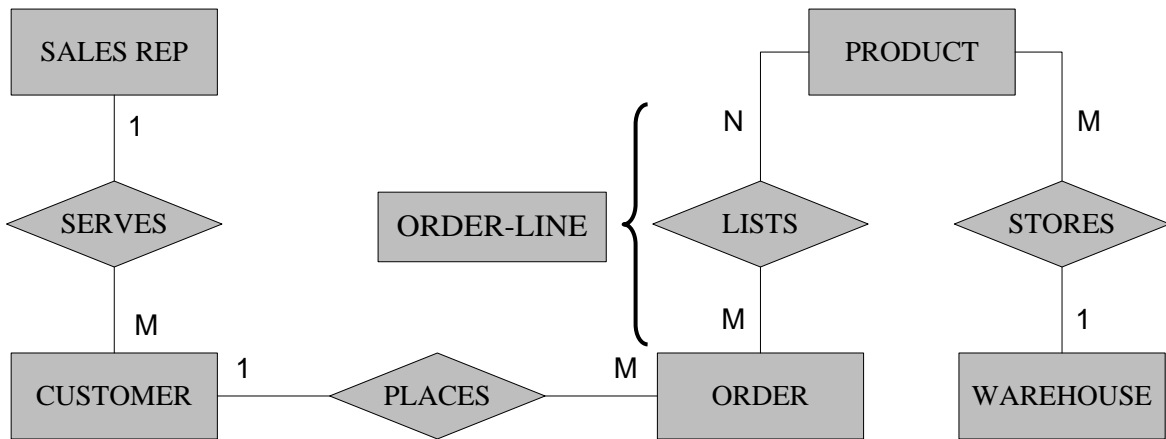
Many-to-many relationship

សរសេរអក្សរកាត់ជា M:N ដែលកើតឡើងនៅពេល record មួយនៅក្នុង entity ទីមួយ ទាក់ទងជា record ជាច្រើននៅក្នុង entity ទីពីរ ហើយ record មួយនៅក្នុង entity ទីពីរ ទាក់ទងជាមួយ record ជាច្រើននៅក្នុង entity ទីមួយ។ ឧទាហរណ៍៖ ទំនាក់ទំនងរវាង STUDENT និង COURSE គឺជា many-to-many មានន័យថា student ម្នាក់អាចសិក្សា courses ជាច្រើន ហើយ course មួយមាន students ជាច្រើនសិក្សា។ រូបខាងក្រោម បង្ហាញពីរូបភាព M:N entity-relationship diagrams ជាច្រើន។ បន្ទាត់មួយតភ្ជាប់ពី entity ទៅកាន់ relationship គេដាក់តួអក្សរ M និងដាក់តួអក្សរ N នៅលើបន្ទាត់មួយផ្សេងទៀត សំរាប់ភ្ជាប់ពី relationship ទៅកាន់ entity មួយទៀតដើម្បីបញ្ជាក់ M:N relationship។

គួរកត់សំគាល់ថា M:N relationship មានលក្ខណៈខុសប្លែកពី 1:1 or 1:M relationship ពីព្រោះព្រឹត្តិការណ៍សំរាប់ភ្ជាប់ entities ទាំង 2 នោះជារឿយៗក្លាយទៅជា entity ទី 3 គេអោយឈ្មោះថា **associative entity** ដែលមានសំនុំ attributes ផ្ទាល់ខ្លួនរបស់វា។ នៅក្នុងរូប 8.9 យើងសង្កេតឃើញថា ENROLL IN relationship បង្ហាញពី REGISTRATION entity ដែលកត់ត្រារាល់ instances ចំពោះ student ជាក់លាក់ណាមួយ ចុះឈ្មោះចូលសិក្សា course ណាមួយ។ ដូចគ្នាផងដែរ RESERVES SEAT ON relationship បង្ហាញពី RESERVATION entity ដែលកត់ត្រារាល់ instances ចំពោះ passenger ជាក់លាក់ណាមួយកក់កៅអីលើជើងយន្តហោះណាមួយ។ LISTS relationship បង្ហាញពី ORDER-LINE entity ដែលកត់ត្រារាល់ instances ចំពោះ product ជាក់លាក់ ណាមួយរាយឈ្មោះនៅក្នុង order ណាមួយ។



(រូប 8.9)



(រូប 8.10)

ERD ពេញលេញនៅក្នុងរូប 8.10 បង្ហាញរាល់ entities and relationships ទាំងអស់ កើតមាននៅក្នុងប្រព័ន្ធ ដោយមាន entities ចំនួន 5 គឺ SALES REP, CUSTOMER, ORDER, PRODUCT, and WAREHOUSE និង relationships រវាង SALES REP and CUSTOMER គឺជា one-to-many relationship ។ ទោះបីជាយ៉ាងណាក៏ដោយ នៅក្នុង

ក្រុមហ៊ុនមួយចំនួនមាន sales rep ជាច្រើនបំរើចំពោះ customer តែម្នាក់ ដូច្នេះនៅក្នុងករណីនេះ យើងត្រូវបំរើជា many-to-many relationship ។

3. 2. Creating an ERD

ការបង្កើត entity-relationship diagrams មានសភាពងាយស្រួល ប្រសិនបើយើងអនុវត្តតាមជំហានទាំង 4 ខាងក្រោម៖

1. *Identify the entities.* ដើម្បីកំណត់ entities យើងត្រូវតែត្រួតពិនិត្យទៅលើ DFDs ដែលយើងបានបង្កើតដោយបង្ហាញពីមនុស្ស, ទីកន្លែង, វត្ថុ វិព្រឹត្តិការណ៍ចំពោះទិន្នន័យបានប្រមូល។ ឧទាហរណ៍៖ ចូរពិចារណាទៅលើ car company យើងសង្កេតឃើញថា មនុស្សមានដូចជា customers and employees, ទីកន្លែងមាន rental locations and return points, វត្ថុមាន vehicle និងព្រឹត្តិការណ៍រួមមាន make a reservation, executing a rental contract, and returning a car ។

2. *Determine all significant events, transactions, or activities that occurs between two or more activities.* កិច្ចការនៅក្នុងជំហាននេះទាមទារអោយយើងវិភាគទៅលើ business operations និងធ្វើការកំណត់ entities ព្រមទាំង relationship and cardinality ផងដែរ។ ឧទាហរណ៍៖ CUSTOMER makes a RESERVATION and a LOCATION has various VEHICLES available for rental ។

3. *Analyze the nature of the interaction.* តើអន្តរកម្មពាក់ព័ន្ធជាមួយ instance មួយនៃ entity រឺ instances ច្រើន? ឧទាហរណ៍៖ ចូរយើងសង្កេតទៅលើទំនាក់ទំនងរវាង CUSTOMER and RESERVATION យើងឃើញថា customer មួយតែម្នាក់គត់អាចគ្មាន reservation រឺមានមួយ រឺមានច្រើន ប៉ុន្តែ reservation នីមួយៗត្រូវមាន customer តែម្នាក់គត់។

4. *Draw the ERD.* យើងអាចគូរ ERD ដោយដៃ រឺប្រើប្រាស់ CASE tool ។ សិក្សាទៅលើ diagrams អោយបានច្បាស់លាស់ និងប្រុងប្រយ័ត្នដើម្បីប្រាកដថា រាល់ entities and relationships បង្ហាញយ៉ាងត្រឹមត្រូវ។

មកដល់ពេលនេះ យើងបានយល់យ៉ាងច្បាស់នូវ database elements and their relationships ដូច្នេះយើងអាចចាប់ផ្តើមកសាង records និងបង្កើតចេញជា data files តែម្តង។ ជំហានជាដំបូងត្រូវដំណើរ normalization ចំពោះ record design ជាមុនសិនដែលរៀបរាប់នៅចំណុចខាងក្រោម។

4. Normalization

Normalization គឺជាដំណើរការធ្វើការកំណត់ និងកែប្រែបញ្ហា និងភាពស្មុគស្មាញជាប់ជាមួយនៅក្នុង record design របស់យើង។ **Record design** ធ្វើការកំណត់ fields និង primary key ចំពោះ records ទាំងអស់នៅក្នុង file ឬ table ណាមួយ។ នៅពេលធ្វើការជាមួយសំណុំ record design ដំបូង យើងត្រូវតែប្រើប្រាស់ normalization ដើម្បីបង្កើត database design ទាំងមូលដែលមានលក្ខណៈសាមញ្ញ (simple), អាចកែប្រែបាន (flexible) និងគ្មាន data redundancy ។

ដំណើរការ normalization ជាទូទៅពាក់ព័ន្ធ 3 ជំហានគឺ: first normal form, second normal form និង third normal form ហើយ normal form ទាំង 3 បង្កើតជា កិច្ចដំណើរការទៅមុខជានិច្ច ធ្វើឃើញណាដើម្បីធ្វើអោយការកសាង database មានលក្ខណៈកាន់តែប្រសើរ ដោយ record design ស្ថិតក្នុង first normal form គ្រាន់តែប្រសើរជាង record design ក្នុង unnormalization ហើយ second normal form កាន់តែប្រសើរជាង first normal form និង third normal form កាន់តែប្រសើរជាង second normal form ។

4. 1. Record Designs

ដំណើរការកសាង records មានលក្ខណៈងាយស្រួល ប្រសិនបើប្រើប្រាស់វិធីស្តង់ដារដើម្បីបង្ហាញពី record structure, fields និង primary keys។ ឧទាហរណ៍នៅផ្នែកខាងក្រោមបង្ហាញពី record design ដោយចាប់ផ្តើមពីឈ្មោះ table ឬ file បន្ទាប់មកសញ្ញាបើករង់ក្រចកតាមដោយឈ្មោះ fields ទាំងអស់ដែលខ័ណ្ឌផ្តាច់ពីគ្នាដោយសញ្ញាចុលភាគ (,) និងបញ្ចប់ដោយសញ្ញាបិទរង់ក្រចកវិញ។ ចំពោះ fields ណាមានតួនាទីជា primary key ត្រូវគូរបន្ទាត់នៅពីខាងក្រោម fields នោះ ហើយ repeating groups of fields ដែលបរិយាយនៅចំនុចខាងក្រោយត្រូវផ្ទុកនៅក្នុងសញ្ញារង់ក្រចក។

NAME (FIELD 1, FIELD 2, FIELD 3, (REPEATING FIELD 4, REPEATING FIELD 5))

4. 2. First Normal Form

Record ស្ថិតក្នុង **first normal form (1NF)** លុះត្រាតែវាគ្មានផ្ទុក repeating group (**repeating group** គឺជាសំណុំ data items ដែលអាចកើតមានជាច្រើនលើកចំពោះ single record)។ យើងអាចគិតថា repeating group គឺជាសំណុំ subsidiary records ផ្ទុកនៅក្នុង main record។ ឧទាហរណ៍: ចូរពិចារណាទៅលើរូបខាងក្រោមចំពោះ ORDER

table យើងសង្កេតឃើញថា records ផ្ទុក product number, description និង number ordered ច្រើនដងច្រើនសារ គេចាត់ទុកជា unnormalized record។ **Unnormalized record** គឺជា record ណាដែលផ្ទុក repeating group មានន័យថា single record មានការបង្ហាញច្រើនលើកចំពោះ fields ណាមួយ ហើយការបង្ហាញនីមួយៗមានតំលៃខុសៗគ្នា។

Primary key		Primary key for repeating group			
<u>ORDER- NUM</u>	<u>ORDER- DATE</u>	<u>PRODUCT- NUM</u>	<u>PRODUCT- DESC</u>	<u>NUM- ORDERED</u>	
40311	03/11/2003	304	Coca Cola	7	} repeating groups
		633	Angkor Beer	1	
		684	ABC Beer	4	
40312	03/11/2003	128	Tiger Beer	12	
		304	Coca Cola	3	
40313	03/12/2003	304	Coca Cola	144	

(រូប 8.11 Unnormalized ORDER table)

ORDER (ORDER-NUM, ORDER-DATE, (PRODUCT-NUM,
PRODUCT-DESC, NUM-ORDERED))

សញ្ញាតំណាងទៅអោយ ORDER record design ខាងលើមានផ្ទុក 5 fields នៅក្នុងសញ្ញាវង់ក្រចកខាងក្រៅ។ ORDER-NUM field មានបន្ទាត់នៅពីខាងក្រោមសំរាប់បញ្ជាក់ថាវាមានតួនាទីជា primary key ហើយ PRODUCT-NUM, PRODUCT-DESC, and NUM-ORDERED fields ស្ថិតនៅក្នុងសញ្ញាវង់ក្រចកខាងក្នុងបញ្ជាក់ថា វាជា fields នៅក្នុង repeating group។ គួរកត់សំគាល់ថា PRODUCT-NUM ក៏មានបន្ទាត់នៅពីក្រោមដែរសំរាប់បញ្ជាក់ពីតួនាទី primary key នៃ repeating group។ ប្រសិនបើអតិថិជនកម្មង់ទិញទំនិញ 3 ប្រភេទខុសៗគ្នានៅក្នុងមួយ order នោះ PRODUCT-NUM, PRODUCT-DESC, and NUM-ORDERED fields ត្រូវបង្ហាញ 3 ដងផងដែរចំពោះមួយ order។

ដើម្បីបំប្លែង unnormalized table ទៅជា 1NF យើងត្រូវកែតម្រូវចំនួន fields នៃ primary key ដោយបញ្ចូលបន្ថែម fields មានតួនាទីជា primary key នៃ repeating group ផងដែរ។ ឧទាហរណ៍៖ repeating group ចំពោះ ORDER table មានចំនួន 3 fields: PRODUCT-NUM, PRODUCT-DESC, and NUM-ORDERED ហើយនៅក្នុងនោះ PRODUCT-NUM ដើរតួនាទីជា primary key ដូច្នេះ primary key នៃ ORDER table

ត្រូវបញ្ចូលបន្ថែម PRODUCT-NUM ជាផ្នែកនៃ primary key នៃ ORDER table ដែរ មានន័យថា primary key (ORDER-NUM, PRODUCT-NUM) ។

ORDER (ORDER-NUM, ORDER-DATE, PRODUCT-NUM,
PRODUCT-DESC, NUM-ORDERED)

ដើម្បីលុប repeating group យើងត្រូវតែបញ្ចូលតំលៃទៅកាន់ cell ណាដែលគ្មានតំលៃ ដោយចំលងតំលៃត្រូវគ្នាជាមួយ record នោះ ហើយលទ្ធផលបង្ហាញដូចរូបខាងក្រោម។

Combination
primary key

<u>ORDER- NUM</u>	ORDER- DATE	<u>PRODUCT- NUM</u>	PRODUCT- DESC	NUM- ORDERED
40311	03/11/2003	304	Coca Cola	7
40311	03/11/2003	633	Angkor Beer	1
40311	03/11/2003	684	ABC Beer	4
40312	03/11/2003	128	Tiger Beer	12
40312	03/11/2003	304	Coca Cola	3
40313	03/12/2003	304	Coca Cola	144

(រូប 8.12 1NF ORDER record)

គួរកត់សំគាល់ថា repeating group ចំពោះ order 40311 ភ្ជាយទៅជា records 3 ផ្សេងៗពីគ្នា ហើយ repeating group ចំពោះ order 40312 ភ្ជាយទៅជា records 2 ផ្សេងៗពីគ្នាផងដែរ។ នៅក្នុង 1NF រាល់ record នីមួយៗរក្សាទុកទិន្នន័យចំពោះទំនិញជាក់លាក់ ណាមួយបានកម្លងទិញលើ order ជាក់លាក់។

Primary key នៃ 1NF មិនអាចជា ORDER-NUM field ទេ ព្រោះ order number មិនអាចធ្វើការកំណត់អត្តសញ្ញាណបាន ដោយសារក្នុងមួយ order មានទំនិញ រឺ items ជាច្រើនប្រភេទ ដូចគ្នាផងដែរ PRODUCT-NUM មិនអាចកំណត់ជា primary key បានដែរ ដោយសារទំនិញមួយត្រូវបានគេកម្លងទិញច្រើនលើកច្រើនសារ។ ហេតុដូច្នេះនេះ ដើម្បីកំណត់ទំនិញណា នៅក្នុង order ណា យើងត្រូវការភាពច្របាច់បញ្ចូលគ្នានៃ ORDER-NUM និង PRODUCT-NUM មកធ្វើការកំណត់អត្តសញ្ញាណ record មានន័យថា ORDER-NUM and PRODUCT-NUM គឺជា primary key ។

4. 3. Second Normal Form

ដើម្បីយល់នូវ second normal form (2NF) យើងត្រូវតែយល់ជាមុននូវ concept of functional dependence ។ Field X is **functional dependent** (អាស្រ័យអនុគមន៍) on

field Y ប្រសិនបើ តំលៃ X ពឹងពាក់ទៅលើតំលៃ Y ឧទាហរណ៍៖ order date ជាអាស្រ័យអនុគមន៍នៃ order number ដោយសារ order date មានតំលៃតែមួយចំពោះ order number នីមួយៗ។

Record design ស្ថិតក្នុង **second normal form (2NF)** ប្រសិនបើវាស្ថិតក្នុង 1NF ហើយរាល់ fields ទាំងអស់ដែលមិនមែនជាផ្នែកនៃ primary key ត្រូវតែជាអាស្រ័យអនុគមន៍នៃ entire primary key ។ ប្រសិនបើ fields ណាមួយស្ថិតក្នុង 1NF ជាអាស្រ័យអនុគមន៍នៃ fields ណាមួយនៃបណ្តុំ primary key មានន័យថា record មិនទាន់ស្ថិតនៅក្នុង 2NF នៅឡើយទេ។ 1NF table ណាមាន primary key កើតពី 1 field ត្រូវស្ថិតក្នុង 2NF ដោយស្វ័យប្រវត្ត។

ចំពោះ ORDER table ខាងលើ យើងសង្កេតឃើញថា primary key កើតឡើងពី ការជុំចូលគ្នានៃ ORDER-NUM និង PRODUCT-NUM ហើយក្នុងនោះ NUM-ORDERED field អាស្រ័យទៅនឹងតំលៃ entire primary key ប៉ុន្តែ ORDER-DATE field អាស្រ័យទៅលើតំលៃ order number តែមួយទេដែលជាផ្នែកនៃ primary key និង PRODUCT-DESC field អាស្រ័យទៅលើ product number តែមួយគត់ដែលជាផ្នែកផ្សេងទៀតនៃ primary key ដូច្នេះ record design មិនទាន់ស្ថិតក្នុង 2NF នៅឡើយទេ។

ហេតុអ្វីបានជាត្រូវការបំប្លែង record design ពី 1NF ទៅ 2NF? ប្រសិនបើយើងមិនបំប្លែងវាទៅជា 2NF វាអាចបង្កបញ្ហាចំនួន 4 ចំពោះ 1NF។ **ទីមួយ** ពិចារណាទៅលើកិច្ចការកែប្រែតំលៃ product's description។ ឧបមាថា យើងមាន 1000 order ចំពោះ product number 304 ដូច្នេះប្រសិនបើចង់កែប្រែ product's description ចំពោះ product number 304 ទាមទារកែប្រែ 1000 records ធ្វើអោយមានលក្ខណៈទើសទែង និងតំលៃថ្លៃ។ **ទីពីរ** 1NF មិនអាចផ្គុំ consistent data។ ប្រសិនបើ product number 304 បង្ហាញចំពោះ 30 order records នោះអាចមាន product descriptions 30 ផ្សេងៗគ្នាចំពោះ product number តែមួយ ដោយគ្មានអ្វីអាចរារាំងមិនអោយវាកើតឡើងបានឡើយ។ **ទីបី** បញ្ហានេះកើតមាននៅពេលយើងចង់បញ្ចូលទិន្នន័យចំពោះទំនិញថ្មីសំរាប់អោយអតិថិជនកម្លង់ទិញ តើត្រូវបញ្ចូលដោយរបៀបណា? យើងដឹងហើយថា ដើម្បីបញ្ចូលតំលៃចំពោះ record នីមួយៗត្រូវបញ្ចូលតំលៃចំពោះ primary key ជាដាច់ខាតគឺ ORDER-NUM និង PRODUCT-NUM ដូច្នេះយើងត្រូវបញ្ចូលអ្វីទៅកាន់ ORDER-NUM នៅពេលយើងចង់បញ្ចូលទំនិញថ្មីមួយមិនទាន់អតិថិជនណាកម្លង់ទិញនៅឡើយ? **ទីបួន** បញ្ហាទាក់ទងនឹងការលុបទិន្នន័យ។ ប្រសិនបើ order ណាបានបង់លុយហើយត្រូវលុបវាចោល ប៉ុន្តែអ្វីនឹងកើតឡើងចំពោះការលុប record ណាផ្ទុកនូវទំនិញមានតែមួយ

ដូចជា product number 633? ប្រសិនបើយើងលុប record នោះធ្វើអោយព័ត៌មានចំពោះ product number and description ត្រូវបាត់បង់។

ដើម្បីបំប្លែង record design ពី 1NF ទៅជា 2NF យើងត្រូវដំណើរការតាមគំរូស្តង់ដារនេះ ដោយជាដំបូង ត្រូវបង្កើត record design ថ្មីចំពោះ field នីមួយៗនៅក្នុង primary key ហើយកំណត់ fields ជា primary key ផង បន្ទាប់មកបង្កើត record design ចំពោះការផ្សំបញ្ចូលគ្នានៃ primary key ដោយម្តងមាន field 2, 3,... ។

ORDER (ORDER-NUM,...)

PRODUCT (PRODUCT-NUM,...)

ORDER-LINE (ORDER-NUM, PRODUCT-NUM,...)

ចុងក្រោយ ដាក់បន្ថែម fields នៅសល់ទៅជាមួយ primary key សាកសមរបស់ពួកវា។

ORDER (ORDER-NUM, ORDER-DATE)

PRODUCT (PRODUCT-NUM, PRODUCT-DESC)

ORDER-LINE (ORDER-NUM, PRODUCT-NUM, NUM-ORDERED)

បន្ទាប់ពីដំណើរការបំប្លែងពី 1NF ទៅជា 2NF យើងទទួលបាន 3 tables ដូចជា៖
ORDER, PRODUCT និង ORDER-LINE ដែលស្ថិតក្នុង 2NF ។

<u>ORDER-NUM</u>	<u>ORDER-DATE</u>
40311	03/11/2003
40312	03/11/2003
40313	03/12/2003

<u>PRODUCT-NUM</u>	<u>PRODUCT-DESC</u>
304	Coca Cola
633	Angkor Beer
684	ABC Beer
128	Tiger Beer

<u>ORDER-NUM</u>	<u>PRODUCT-NUM</u>	<u>NUM-ORDERED</u>
40311	304	7
40311	633	1
40311	684	4
40312	128	12
40312	304	3
40313	304	144

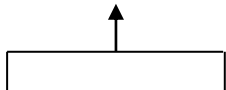
(រូប 8.13 2NF record)

4. 4. Third Normal Form

នៅពេលរាល់ tables ទាំងអស់ស្ថិតក្នុង 3NF វាបានជៀសវាង redundancy and data integrity problems ដែលអ្នកកសាង database តែងតែអនុវត្តតាម។ Table ស្ថិតក្នុង 2NF ក៏បង្កបញ្ហាដូចទៅនឹង 1NF ដែរ។

Record design ស្ថិតក្នុង **third normal form (3NF)** ប្រសិនបើវាស្ថិតក្នុង 2NF ហើយគ្មាន *nonkey field* ជាអាស្រ័យអនុគមន៍នៃ nonkey field ផ្សេងទៀតទេ។ គួរចងចាំថា nonkey field មិនមែនជា candidate key ទេ។ រូបខាងក្រោមបង្ហាញពី CUTOMER table ស្ថិតនៅក្នុង 2NF ព្រោះមាន nonkey field មួយគឺ SALES-REP-NAME ជាអាស្រ័យអនុគមន៍ nonkey field ផ្សេងទៀតគឺ SALES-REP-NUM ។

Nonkey field SALES-REP-NAME ជាអាស្រ័យអនុគមន៍ នៃ nonkey field ផ្សេងទៀតគឺ SALES-REP-NUM



<u>CUSTOMER- NUM</u>	CUSTOMER- NAME	ADDRESS	SALES- REP-NUM	SALES- REP-NAME
108	Benedict Luis	Diego, CA	41	Kap James
233	Corelli Helen	Nashua, NH	22	McBride Jon
254	Geomez J.P	Butte, MT	38	Stein Ellen
431	Lee M.	Camp, NC	74	Roman Rold
779	Paulski Diane	Lead, SD	38	Stein Ellen
800	Zuider Z.	Greer, SC	74	Roman Rold

(រូប 8.14 2NF record design)

ដើម្បីបំប្លែង record design ទៅជា 3NF យើងត្រូវតែលុប fields ទាំងឡាយណា ជាអាស្រ័យអនុគមន៍នៃ nonkey field ផ្សេងទៀត ហើយដាក់ពួកវាទៅកាន់ table ថ្មីដោយកំណត់ nonkey field ដែលគេពឹងពាក់ជា primary key ។ យើងបំប្លែង CUSTOMER table ទៅជា tables ស្ថិតក្នុង 3NF ដោយលុប SALES-REP-NAME field ចេញពី CUSTOMER table ហើយផ្ទុកវាទៅកាន់ table ថ្មី និងកំណត់ SALES-REP-NUM ជា primary key ។

CUSTOMER (CUSTOMER-NUM, CUSTOMER-NAME, ADDRESS,
SALES-REP-NUM)

SALES-REP (SALES-REP-NUM, SALES-REP-NAME)

<u>CUSTOMER- NUM</u>	CUSTOMER- NAME	ADDRESS	SALES- REP-NUM
108	Benedict Luis	Diego, CA	41
233	Corelli Helen	Nashua, NH	22
254	Geomez J.P	Butte, MT	38
431	Lee M.	Camp, NC	74
779	Paulski Diane	Lead, SD	38
800	Zuider Z.	Greer, SC	74

<u>SALES-REP- NUM</u>	SALES-REP- NAME
41	Kap James
22	McBride Jon
38	Stein Ellen
74	Roman Rold

(រូប 8.15)

5. Steps in Database Design

យើងអាចចាប់ផ្តើមបង្កើត database បន្ទាប់ពីយើងបានដំណើរការ normalization ទៅលើ record design ដោយដំណើរការទៅតាម analysis and design steps ទាំង 4 ដូចខាងក្រោម:

1. *Create the initial ERD.* ចាប់ផ្តើមត្រួតពិនិត្យឡើងវិញទៅលើ DFDs និង class diagrams ដើម្បីកំណត់ system entities ។ លើសពីនេះទៀត យើងគួរតែពិចារណាទៅលើ data stores បង្ហាញលើ DFDs ដើម្បីកំណត់ថាតើវាគួរបង្ហាញជា entities ដែររឺទេ? បន្ទាប់មកវិភាគថា តើពួកវាទំនាក់ទំនងក្នុងទម្រង់ណាមួយ 1:1, 1:M or M:N? ។ ឧទាហរណ៍: បង្កើត ERD ចំពោះ entities MEMBER និង VIDEO នៅក្នុង video rental system ។

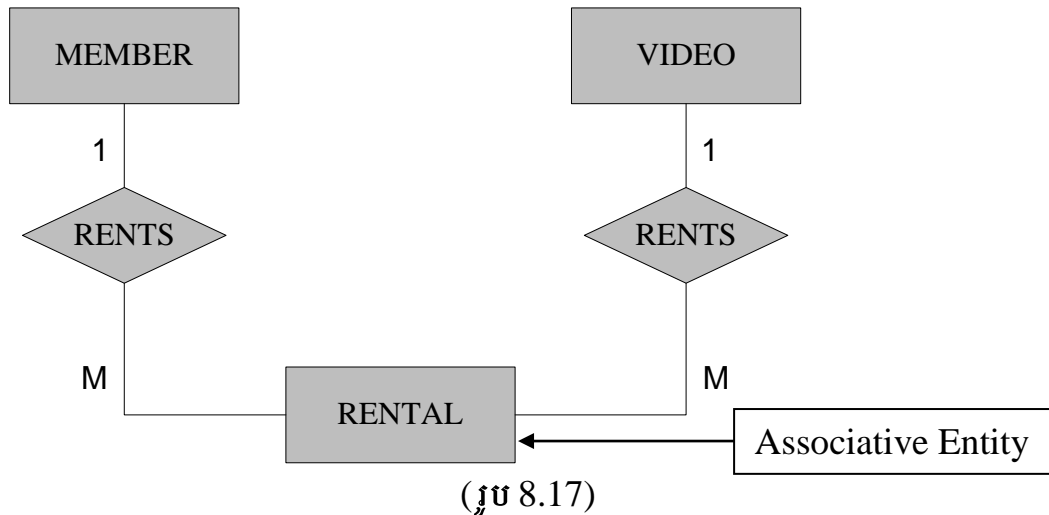


(រូប 8.16)

2. *Assign all data elements to entities.* មើលបញ្ជាក់គ្រប់ data element ទាំងអស់នៅក្នុង data dictionary ដែលពាក់ព័ន្ធជាមួយ entity ។

MEMBER (MEMBER-NUMBER, NAME, ADDRESS, PHONE,
 (VIDEO-ID, TITLE, DATE-RENTED, DATE-RETURNED))
 VIDEO (VIDEO-ID, TITLE)

3. Create 3NF designs for all records, taking care to identify all primary, secondary, and foreign keys. បង្កើត ERD ចុងក្រោយដោយមានបញ្ចូល entities ថ្មីៗដែលបានកំណត់កំឡុងពេល normalization ។



MEMBER (MEMBER-NUMBER, NAME, ADDRESS, PHONE)
 VIDEO (VIDEO-ID, TITLE)
 RENTAL (MEMBER-NUMBER, VIDEO-ID, DATE-RENTED,
 DATE-RETURNED)

4. Verify all data dictionary entries. ធ្វើអោយប្រាកដថា យើងបានកត់ត្រាជាឯកសារយ៉ាងត្រឹមត្រូវ និងពេញលេញរាល់ data dictionary entries ចំពោះ data stores, records និង data elements ។

បន្ទាប់ពីបង្កើត ERD ចុងក្រោយ និងដំណើរការ normalization ទៅលើ record designs រួចហើយ យើងអាចបំប្លែងវាទៅជា database ។

Review Questions

1. ចូរពន្យល់ពីភាពខុសគ្នារវាង file-oriented system និង database system?
2. ចូរអោយនិយមន័យពាក្យ: primary key, candidate key, foreign key?
3. ចូររៀបរាប់ពីបញ្ហាទាំង 3 ដែលកើតមានក្នុង file processing?
4. ចូររៀបរាប់ពី advantages of DBMS?
5. អ្វីទៅជា entity-relationship diagrams និងតើគេប្រើប្រាស់វានៅពេលណា?
6. អ្វីទៅជា relationship? ចែកជាប៉ុន្មាន? ចូរពន្យល់។
7. អ្វីទៅជា normalization? តើគេចែកជាប៉ុន្មានជំហាន? ចូរពន្យល់ពីជំហាននីមួយៗ?
8. ចូរពន្យល់ពី steps in database design?

