



ສາທາລະນະລັດ ປະຊາທິປະໄຕ ປະຊາຊົນລາວ  
ສັນຕິພາບ ເອກະລາດ ປະຊາທິປະໄຕ ເອກະພາບ ວັດທະນາຖາວອນ

-----((0))-----



ມະຫາວິທະຍາໄລແຫ່ງຊາດ  
ຄະນະວິທະຍາສາດທຳມະຊາດ  
ພາກວິຊາວິທະຍາສາດຄອມພິວເຕີ

ບົດລາຍງານ ວິຊາ ວິທີການຄົ້ນຄວ້າ  
ສາຂາ (ຕໍ່ເນື່ອງ) ວິທະຍາສາດຄອມພິວເຕີ

ຊື່ບົດຈົບຊັ້ນ (Title)

ພາສາລາວ: ລະບົບຈອງປີ້ລົດເມສາຍໃຕ້ອອນລາຍ

ພາສາອັງກິດ: Southern Bus Ticket Online Booking System

ສະມາຊິກໃນກຸ່ມ (Project Team)

ລ/ດ	ລະຫັດ	ຊື່ ແລະ ນາມສະກຸນ	ເບີໂທ
1	204N0025.19	ທ ພອນຄຳ ແກ້ວມະນີ	020 28022677
2	204N0002.19	ທ ມະໂນພອນ ມະໂນກຸນ	020 54000003

ສອນໂດຍ ປອ. ລັດສະໝີ ຈິດຕະວົງ

## 1. ຄວາມສໍາຄັນຂອງບັນຫາ

ໃນຍຸກປັດຈຸບັນເຕັກໂນໂລຊີຂໍ້ມູນຂ່າວສານມີຄວາມກ້າວໜ້າ ແລະ ທັນສະໄໝຫຼາຍຂຶ້ນເປັນຕົ້ນແມ່ນເຄືອຂ່າຍ Internet ໄດ້ຂະຫຍາຍຕົວຢ່າງວ່ອງໄວຈົນກ້າວເຂົ້າສູ່ບົດບາດ ແລະ ຄວາມສໍາຄັນຢ່າງຫຼວງຫຼາຍ. ເຮັດໃຫ້ມີການຄິດຄົ້ນສິ່ງອໍານວຍຄວາມສະດວກເພື່ອພັດທະນານຳໃຊ້ໃນຊີວິດປະຈຳວັນຂອງພວກເຮົາ ບໍ່ວ່າຈະເປັນດ້ານການຕິດຕໍ່ສື່ສານ, ການປະຊາສຳພັນຂ່າວ, ການໂຄສະນາ, ການສຶກສາ, ສື່ສານຕ່າງໆ ແລະ ຕະຫຼອດຮອດການຄ້າຂາຍເປັນຕົ້ນກໍ່ໄດ້ນຳເອົາເຕັກໂນໂລຊີເຂົ້າມາຊ່ວຍໃນການດຳເນີນງານໃຫ້ສະດວກສະບາຍວ່ອງໄວ, ນອກຈາກນັ້ນຍັງສາມາດເຜີຍແຜ່ຂໍ້ມູນຂ່າວສານຕ່າງໆຜ່ານເວັບໄຊຕ່າງໆ. ເຮັດໃຫ້ເວັບໄຊໃນເຄືອຂ່າຍໃນອິນເຕີເນັດສາມາດຕອບສະໜອງໃນດ້ານຕ່າງໆຜ່ານເວັບໄຊຕ່າງໆ. ເຮັດໃຫ້ເວັບໄຊໃນເຄືອຂ່າຍໃນອິນເຕີເນັດສາມາດຕອບສະໜອງໃນດ້ານຕ່າງໆທີ່ກ່າວມານັ້ນເປັນຢ່າງດີ ແລະ ຈະສັງເກດໄດ້ດ້ວຍໜ່ວຍງານອົງກອນທຸລະກິດ, ບໍລິສັດ ຫຼື ຮ້ານຄ້າໃຫຍ່ຕ່າງໆ ຈະມີຄວາມສົນໃຈ ແລະ ແນໃສ່ຄວາມສໍາຄັນຂອງສິນຄ້າຫຼາຍຂຶ້ນ ຈຸດປະສົງເພື່ອປະຊາສຳພັນ ຫຼື ການຄ້າຂາຍ, ການສ້າງເວບໄຊຂອງບໍລິສັດ ເພື່ອບໍລິຫານການຂາຍສິນຄ້າກໍ່ຖືວ່າເປັນທາງເລືອກໜຶ່ງທີ່ທັນສະໄໝ ເປັນການອໍານວຍຄວາມສະດວກໃຫ້ແກ່ພະນັກງານ ແລະ ເຈົ້າຂອງບໍລິສັດ, ເປັນການໂຄສະນາບໍລິສັດຜ່ານທາງເວບໄຊໃຫ້ເປັນທີ່ຮູ້ຈັກກັນຫຼາຍຂຶ້ນ ເປັນທາງເລືອກໜຶ່ງທີ່ສະດວກສະບາຍໃນການບໍລິຫານການຂາຍທີ່ລູກຄ້າສາມາດເລືອກຊື້ ແລະ ເບິ່ງສິນຄ້າພາຍໃນບໍລິສັດໄດ້.

ສະຖານີຂົນສົ່ງໂດຍສານສາຍໃຕ້ ເປັນບ່ອນບໍລິການຮັບ-ສົ່ງຜູ້ໂດຍສານ, ສິນຄ້າວັດຖຸສິ່ງຂອງ ແລະ ສັດ ຈາກຈຸດໜຶ່ງໄປຫາອີກຈຸດໜຶ່ງ ຊຶ່ງສະຖານີຂົນສົ່ງໂດຍສານທາງໄກສາຍໃຕ້ນີ້ແມ່ນໄດ້ ສ້າງຕັ້ງຂຶ້ນໃນວັນທີ 1 ກັນຍາ 2016 ເຊິ່ງມີລາຍລະອຽດດັ່ງລຸ່ມນີ້: ທີ່ຕັ້ງ ແລະ ພາລະບົດບາດຂອງສະຖານີຂົນສົ່ງທາງໄກສາຍໃຕ້ ແມ່ນສະຖານີໜຶ່ງຊຶ່ງຕັ້ງຢູ່ບ້ານສະພັງມຶກ, ເມືອງ ໄຊທານີ, ນະຄອນຫຼວງວຽງຈັນ, ຖະໜົນເລກທີ 450 ປີ ໃກ້ກັບ ສີ່ແຍກໄຟແດງດົງໂດກ. ສະຖານີຂົນສົ່ງທາງໄກສາຍໃຕ້ ປະກອບມີຫຼາຍໜ່ວຍງານຄື: ອໍານວຍການໃຫຍ່ມີ 1 ທ່ານ, ເລຂານຸການມີ 1 ທ່ານ, ໜ່ວຍງານແຜນການມີ 1 ທ່ານ, ໜ່ວຍງານຮັບ-ຈ່າຍເງິນມີ 1 ທ່ານ, ໜ່ວຍງານບໍລິການຂາຍປີ້ມີ 7 ທ່ານ, ໜ່ວຍງານຮັກສາຄວາມປອດໄພມີ 6 ທ່ານ, ໜ່ວຍງານບໍລິການເຮືອນພັກມີ 8 ທ່ານ ແລະ ບັນດາບໍລິສັດ ທີ່ເຂົ້າມາດຳ ເນີນທຸລະກິດ ໃນສະຖານີຂົນສົ່ງໂດຍສານປະກອບມີ: ບໍລິສັດ ຂົນສົ່ງໂດຍສານຈົດປະສົງ ຍອດນິຍົມ, ບໍລິສັດ ແສງສົມ ບູນ ຂົນສົ່ງໂດຍສານ, ບໍລິສັດ ແສນສະບາຍ ຂົນສົ່ງໂດຍສານ, ບໍລິສັດ ຈັນທະຈອນ ຂົນສົ່ງໂດຍສານ, ບໍລິສັດ ຈຳປາສັກ ຂົນສົ່ງໂດຍສານ, ບໍລິສັດ ແສງຈະເລີນ ລົດຕຽງນອນ, ບໍລິສັດ ກຽງໄກ VIP, ບໍລິສັດ ສີທອນ ພວງປະເສີດ ລົດຕຽງນອນ. ນອກຈາກນີ້ ສະຖານີຍັງ ມີສະຖານທີ່ພັກ , ຮ້ານຄ້າ, ຮ້ານຂາຍຍ່ອຍ, ຮ້ານອາຫານ ແລະ ສິ່ງອໍານວຍຄວາມສະດວກຕ່າງໆ ໄວ້ເພື່ອບໍລິການຜູ້ໂດຍສານທີ່ມາລໍຖ້າລົດໄປຈຸດໝາຍປາຍທາງ.

ເນື່ອງຈາກວ່າ ຈັດການຂໍ້ມູນ, ລາຍງານຂໍ້ມູນຕ່າງໆ, ລວມທັງການຂາຍປີ້ແມ່ນຍັງໃຊ້ແບບຈົດ ແລະ ຜູ້ໂດຍສານທີ່ຕ້ອງການຈອງປີ້ລວງໜ້າ ຕ້ອງໄດ້ໂທຫາພະນັກງານຂາຍປີ້ເພື່ອຈອງເຊິ່ງເຮັດໃຫ້ການບໍລິການມີການຊັກຊ້າ ແລະ ຂໍ້ມູນຍັງມີການຕົກເຮ່ຍເສຍຫາຍ

ດັ່ງນັ້ນ , ພວກຂ້າພະເຈົ້າຈຶ່ງເຫັນຄວາມສໍາຄັນຂອງບັນຫາ ຈຶ່ງມີແນວຄວາມຄິດທີ່ຈະສ້າງລະບົບຈອງປີ້ລົດເມແບບອອນໄລ ຂອງສະຖານີຂົນສົ່ງໂດຍສານສາຍໃຕ້ນັ້ນຂຶ້ນມາ ເພື່ອຊ່ວຍຫຼຸດຜ່ອນຄວາມຫຍຸ້ງຍາກໃນການຈອງປີ້ລົດ, ຈັດເກັບຂໍ້ມູນ, ຫຼຸດຜ່ອນຄວາມຊັກຊ້າໃນການຈັດການຂໍ້ມູນ, ເຮັດໃຫ້ຂໍ້ມູນມີຄວາມເປັນລະບຽບຮຽບຮ້ອຍ ແລະ ເພື່ອໃຫ້ມີຄວາມສະດວກວ່າອ່ງໄວຕໍ່ການຄົ້ນຫາຂໍ້ມູນ.

## **2. ຈຸດປະສົງຂອງການຄົ້ນຄວ້າ (Objectives)**

- ເພື່ອສຶກສາບັນຫາທີ່ເກີດຂຶ້ນໃນປະຈຸບັນ ແລະ ຄວາມຕ້ອງການຂອງລະບົບ.
- ເພື່ອສ້າງລະບົບຂາຍປີ້ລົດອອນໄລຂອງສະຖານີຂົນສົ່ງໂດຍສານສາຍໃຕ້.
- ເພື່ອສ້າງຮູບແບບການຈັດການຂໍ້ມູນການໃຫ້ບໍລິການ.
- ເພື່ອເຜີຍແຜ່ຂໍ້ມູນການຂາຍປີ້ລົດເມຂອງສະຖານີຂົນສົ່ງໂດຍສານສາຍໃຕ້.
- ເພື່ອການລາຍງານໃຫ້ສະດວກ ແລະ ຖືກຕ້ອງ.

## **3. ຂອບເຂດການຄົ້ນຄວ້າ (Scope)**

ລະບົບຈອງປີ້ລົດເມສາຍໃຕ້ແບບອອນລາຍ ເປັນລະບົບແບບ Client-Server Web Application ເຊິ່ງປະກອບດ້ວຍໜ້າວຽກໜັກດັ່ງນີ້:

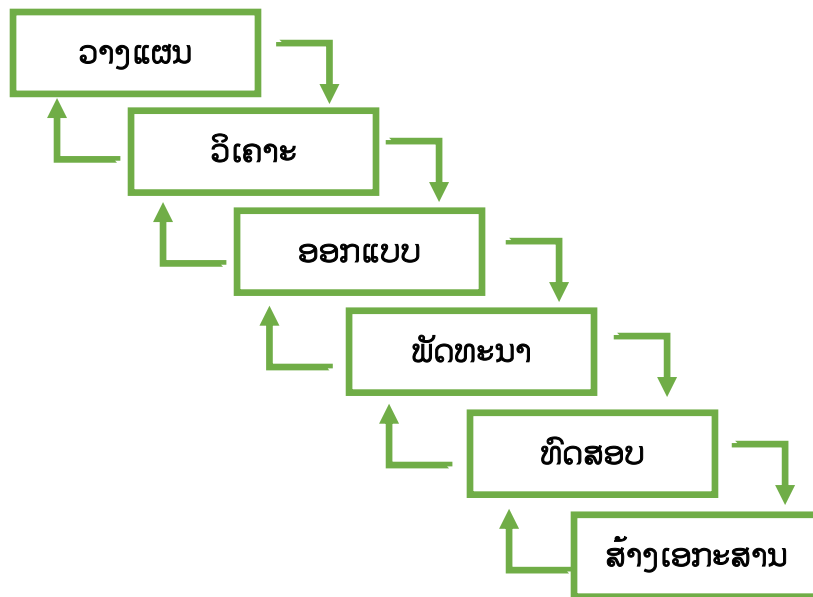
- ຈັດການຂໍ້ມູນພື້ນຖານ : (ຂໍ້ມູນພະນັກງານ, ຂໍ້ມູນລົດ, ຂໍ້ມູນປະເພດລົດ, ຂໍ້ມູນສາຍທາງ)
- ສະໝັກສະມາຊິກ
- ບໍລິການ : (ຈອງປີ້, ອອກປີ້)
- ລາຍງານ : (ລາຍງານຂໍ້ມູນການຈອງ, ລາຍງານຂໍ້ມູນພະນັກງານ, ລາຍງານຂໍ້ມູນສາຍທາງ, ລາຍງານຂໍ້ມູນລົດ, ລາຍງານຂໍ້ມູນອອກປີ້)

## **4. ປະໂຫຍດຄາດວ່າຈະໄດ້ (Expected Outcome of the Project)**

- ໄດ້ລະບົບຈອງປີ້ລົດເມແບບອອນລາຍຂອງສະຖານີຂົນສົ່ງສາຍໃຕ້
- ໄດ້ລະບົບທີ່ຈະຊ່ວຍແກ້ໄຂບັນຫາການຈອງ ຈອງໄດ້ສະດວກ ແລະ ວ່ອງໄວຂຶ້ນ
- ໄດ້ລະບົບຊ່ວຍເພີ່ມຊ່ອງທາງໃນການຂາຍປີ້ໃຫ້ກັບຜູ້ປະກອບການ
- ມີລະບົບເຜີຍແຜ່
- ໄດ້ລະບົບທີ່ສາມາດສ້າງລາຍງານໄດ້ຢ່າງສະດວກ ແລະ ຖືກຕ້ອງ

## **5. ວິທີດໍາເນີນການຄົ້ນຄວ້າ (Research Methodology)**

ຂັ້ນຕອນວິທີການດໍາເນີນການພັດທະນາລະບົບໃນຄັ້ງນີ້ແມ່ນອີງໃສ່ລັກສະນະແບບຈໍາລອງຂອງ Adapted Waterfall Model ເປັນແນວທາງໃນການວິເຄາະ ແລະ ອອກແບບລະບົບເຊິ່ງມີໜ້າວຽກດັ່ງນີ້:



ຮູບທີ່ 1 ວົງຈອນການພັດທະນາແບບນ້ຳຕົກ (Adapted Waterfall Model)

- **ໄລຍະການວາງແຜນ**

ດຳເນີນການຈັດຕັ້ງກຸ່ມຂຽນບົດໂຄງການພຽງພ້ອມກັບການກຳນົດຂໍ້ຂອງໂຄງການກຳນົດຫົວຂໍ້ຂອງໂຄງການ. ຈາກນັ້ນ, ກໍໄດ້ລົງເກັບກຳຂໍ້ມູນຢູ່ສະຖານນີຂົນສົ່ງໂດຍສານສາຍໃຕ້ ເພື່ອໃຫ້ໄດ້ຂໍ້ມູນ ແລະ ຂັ້ນຕອນການເຮັດວຽກໂດຍລວມກ່ຽວກັບການເຮັດວຽກຕ່າງໆຂອງສະຖານີດັ່ງກ່າວ.

- **ໄລຍະການວິເຄາະ**

ໄລຍະນີ້ພວກເຮົາຈະສຶກສາລະບົບເກົ່າ ແລະ ຄວາມຕ້ອງການຂອງຜູ້ໃຊ້. ຈາກນັ້ນ, ກໍນຳເອົາຂໍ້ມູນຕ່າງໆລວບລວມເອົາຂໍ້ມູນຕ່າງໆທີ່ລວບລວມໄດ້ມາເພື່ອວິເຄາະເປັນຂໍ້ກຳນົດຄວາມຕ້ອງການຂອງລະບົບໃໝ່ພ້ອມທັງແຕ້ມແບບຈຳລອງ DFD ເພື່ອສະແດງເຖິງການໄຫຼຂໍ້ມູນໄປເຖິງຂະບວນການ ແລະ ແຜນວາດ E-R Diagram ເພື່ອສະແດງຄວາມສຳພັນລະຫວ່າງຂໍ້ມູນ.

- **ໄລຍະການອອກແບບ**

ໄລຍະການອອກແບບ ແມ່ນໄລຍະທີ່ພວກຂ້າພະເຈົ້າ ນຳເອົາຜົນທີ່ໄດ້ຈາກການວິເຄາະມາອອກແບບລະບົບໂດຍການອອກແບບ, ໜ້າຟອມປ້ອນຂໍ້ມູນ ແລະ ໜ້າຟອມລາຍງານໂດຍນຳໃຊ້ Visual Studio code ລວມທັງອອກແບບຖານຂໍ້ມູນ.

- **ໄລຍະການພັດທະນາ**

ດຳເນີນການສ້າງຂໍ້ມູນຕາມທີ່ໄດ້ເຮັດ Data Dictionary ໃນຂັ້ນຕອນກ່ອນໜ້ານັ້ນ. ຈາກນັ້ນ, ກໍລົງມືປະຕິບັດຂຽນ Code ເພື່ອຕິດຕໍ່ ແລະ ເຮັດວຽກກັບຖານຂໍ້ມູນທີ່ໄດ້ສ້າງໄວ້ແລ້ວ.

- ໂລຍະການທົດສອບ

ສ້າງແບບວິທີການທົດສອບລະບົບ ແລະ ນຳເອົາຊຸດຂໍ້ມູນທີ່ນຳມາທົດສອບຢ່າງໜ້ອຍ 20 records ມາດຳເນີນການທົດສອບແລ້ວສະຫຼຸບຜົນໄດ້ຮັບ ແລະ ທຳການປັບປຸງແກ້ໄຂຈົນໄດ້ຜົນອອກທີ່ຖືກຕ້ອງ

- ໂລຍະການສ້າງເອກະສານ

ສ້າງປຶ້ມບົດຈົບຊັ້ນ ລະບົບການຈອງປີ້ແບບອອນລາຍຂອງສະຖານີຂົນສົ່ງໂດຍສານສາຍໄຕ້ ແລ້ວສົ່ງມອບປຶ້ມພ້ອມທັງແຜ່ນ CD ທີ່ປະກອບດ້ວຍ Source Code ,ເອກະສານບົດຈົບຊັ້ນ ແລະ Presentations.

## 6. ສະຖານທີ່ສຶກສາ (Study Site)

ສະຖານີຂົນສົ່ງ ໂດຍສານວຽງຈັນ ຈຳກັດ (ສາຍໄຕ້-ຕ່າງປະເທດ)

## 7. ໂລຍະປະຕິບັດ (Duration)



## 8. ເຄື່ອງມືທີ່ນຳໃຊ້ໃນການພັດທະນາ (Development Tools)

1. Hardware	2. Software
<ul style="list-style-type: none"> <li>➢ ຄອມພິວເຕີ Lenovo Intel(R) Core(TM) I5-3230M CPU @2.60GHz, RAM 8GB DDR3L 1600MHz, SSD 240GB,HDD 500GB</li> <li>➢ USB 16GB 1ອັນ</li> </ul>	<ul style="list-style-type: none"> <li>➢ ລະບົບປະຕິບັດການ Windows 10 Professional 64 Bit</li> <li>➢ Microsoft Visio 2016 ໃຊ້ແຕ້ມແຜນວາດການໄຫຼຂໍ້ມູນ (DFD, ER, Flowchart)</li> <li>➢ Adobe XD ໃຊ້ອອກແບບ UX/UI</li> <li>➢ Studio 3T For MongoDB ແລະ Moon Modeler ໃຊ້ຈັດການຖານຂໍ້ມູນ ແລະ ອອກແບບ Database Model</li> <li>➢ Visual Studio Code ໃຊ້ຂຽນໂຄດດ້ວຍພາສາ JavaScript (ReactJS, NodeJS, GraphQL)</li> </ul>

	➢ MS Office 2013 Professional ໃຊ້ເພື່ອສ້າງເອກະສານຕ່າງໆ, ບົດນຳສະເໜີ ແລະ ເຮັດປຶ້ມບົດຈົບຊັ້ນ
--	---

## 9. ເອກະສານອ້າງອີງ (Reference)

ສົມມິດ ທຸມມາລີ, ແລະ ອາມອນ ຈັນທະພາວົງ. (2012). ການວິເຄາະ ແລະ ອອກແບບລະບົບ (System Analysis and Design), ໜ້າ 56, 68 ນະຄອນຫຼວງວຽງຈັນ: ໂຮງພິມນະຄອນຫຼວງວຽງຈັນ

## 10. ທົບທວນທິດສະດີ ແລະ ບົດທີ່ກ່ຽວຂ້ອງ

### 10.1. ທົບທວນທິດສະດີທີ່ກ່ຽວຂ້ອງ

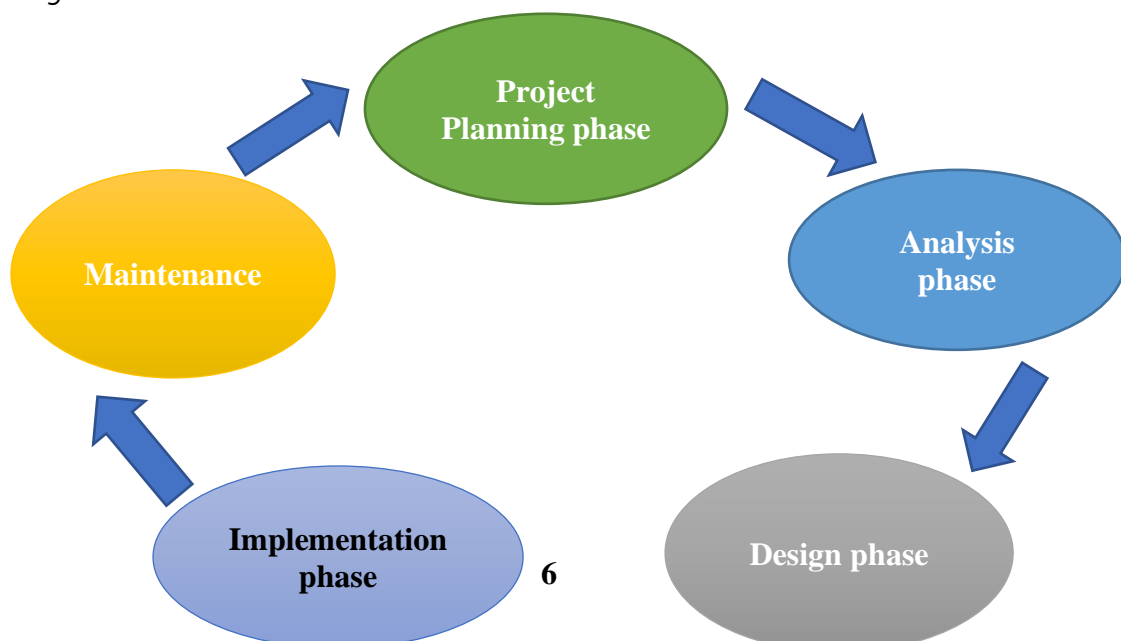
ລະບົບຈອງປຶ້ມລົດເມສາຍໃຕ້ອອນລາຍແມ່ນພັດທະນາຂຶ້ນໂດຍນຳໃຊ້ທິດສະດີ ຫຼື ຄວາມຮູ້ຈາກ 3 ສ່ວນຄື: ທິດສະດີໃນການວິເຄາະ ແລະ ອອກແບບລະບົບ, ທິດສະດີກ່ຽວກັບການສ້າງຖານຂໍ້ມູນ ແລະ ຄວາມຮູ້ກ່ຽວກັບພາສາໃນການພັດທະນາລະບົບ. ລາຍລະອຽດຂອງແຕ່ລະທິດສະດີແມ່ນຈະໄດ້ນຳສະເໜີໂດຍສັງເຂບດັ່ງຕໍ່ໄປນີ້:

#### 10.1.1. ທິດສະດີກ່ຽວກັບການວິເຄາະ ແລະ ການອອກແບບລະບົບ

ການວິເຄາະລະບົບເປັນຂະບວນການທຳຄວາມເຂົ້າໃຈ ແລະ ກຳນົດລາຍລະອຽດຂອງບັນຫາເພື່ອຈະໄດ້ພິຈາລະນານຳເອົາລະບົບຂໍ້ມູນຂ່າວສານໃດເຂົ້າໄປແກ້ບັນຫາເຫຼົ່ານັ້ນ, ສ່ວນການອອກແບບລະບົບໝາຍເຖິງຂະບວນການກຳນົດລາຍລະອຽດຕ່າງໆວ່າຈະຕ້ອງເຮັດແນວໃດກັບອົງປະກອບຂອງລະບົບຂໍ້ມູນຂ່າວສານເພື່ອຈະໄດ້ນຳໄປໃຊ້ໃຫ້ເກີດຜົນໃນທາງພາຍນອກໄດ້. ສິ່ງທີ່ນຳສະເໜີໃນຫົວຂໍ້ນີ້ໄດ້ຂັດສັນພາກສ່ວນໜຶ່ງຈາກ (ພຣະ ເຫຼົາຄຳ ເພັດວິໄລ, ວິລະຍຸດ ວົງທິລາດ ແລະ ເພັດດາວວອນ ທິແກ້ວ, 2016) ແລະ (ສົມມິດ ທຸມມາລີ ແລະ ອາມອນ ຈັນທະພາວົງ, 2012).

##### 10.1.1.1. ວົງຈອນພັດທະນາລະບົບ

ເປັນຂະບວນການທີ່ສະແດງເຖິງການດຳເນີນຂັ້ນຕອນການເຮັດວຽກຂອງລະບົບຕັ້ງແຕ່ຕົ້ນຈົນຈົບ, ບົດໂຄງການຈົບຊັ້ນນີ້ພວກຂ້າພະເຈົ້າໄດ້ນຳໃຊ້ທິດສະດີການວິເຄາະ ແລະ ອອກແບບລະບົບແບບໂຄງທີ່ປະກອບມີ 5 ໄລຍະຄື:





- ໄລຍະທີ 1 ການວາງແຜນໂຄງການ (Project Planning phase)
- ໄລຍະທີ 2 ການວິເຄາະ (Analysis phase)
- ໄລຍະທີ 3 ການອອກແບບ (Design phase)
- ໄລຍະທີ 4 ການນຳໄປໄຊ້ (Implementation phase)
- ໄລຍະທີ 5 ການບຳລຸງຮັກສາ (Maintenance phase)

#### **ກ. ໄລຍະທີ 1 ການວາງແຜນໂຄງການ (Project planning phase)**

ການວາງແຜນໂຄງການຈັດເປັນຂະບວນການພື້ນຖານໃນຄວາມເຂົ້າໃຈຢ່າງເລິກເຊິ່ງວ່າເປັນ ຫຍັງຕ້ອງສ້າງລະບົບໃໝ່ ທີ່ມີງານຕ້ອງພິຈາລະນາວ່າ ຈະຕ້ອງດຳເນີນງານຕໍ່ໄປແນວໃດກ່ຽວກັບ ຂະບວນການສ້າງລະບົບໃໝ່ ກ່ອນອື່ນໝົດຕ້ອງມີຈຸດກຳເນີດຂອງລະບົບງານເຊິ່ງໂດຍປົກກະຕິແລ້ວ ຈຸດກຳເນີດຂອງລະບົບງານມັກເກີດຂຶ້ນຈາກຜູ້ໃຊ້ລະບົບເປັນຜູ້ທີ່ຕິດແທດກັບລະບົບໂດຍກົງເຮັດໃຫ້ມີ ຄວາມໃກ້ສືດກັບລະບົບງານທີ່ດຳເນີນຢູ່ຫຼາຍທີ່ສຸດເມື່ອຜູ້ໃຊ້ລະບົບມີຄວາມຕ້ອງການປັບປຸງລະບົບ ງານ ດັ່ງນັ້ນຈຶ່ງຖືເປັນຈຸດເລີ່ມຕົ້ນໃນບົດບາດຂອງນັກວິເຄາະລະບົບວ່າຈະຕ້ອງສຶກສາເຖິງຂອບເຂດ ຂອງບັນຫາທີ່ຜູ້ໃຊ້ລະບົບກຳລັງປະສົບບັນຫາຢູ່ ແລະ ຈະດຳເນີນການແກ້ໄຂແນວໃດ ສຶກສາເຖິງ ຄວາມເປັນໄປໄດ້ວ່າລະບົບໃໝ່ທີ່ຈະພັດທະນາຂຶ້ນມານັ້ນມີຄວາມເປັນໄປໄດ້ ແລະ ຄຸ້ມຄ່າກັບການຈະ ລົງທຶນ ຫຼື ບໍ່.

ແນວໃດກໍຕາມໄລຍະຂອງການວາງແຜນໂຄງການປົກກະຕິມັກຈະມີໄລຍະເວລາທີ່ສັ້ນ ແຕ່ກໍຖື ວ່າເປັນຂັ້ນຕອນທີ່ສຳຄັນທີ່ຈະໃຫ້ເກີດຜົນສຳເລັດໄດ້ ດັ່ງນັ້ນໃນໄລຍະຂອງການວາງແຜນໂຄງການຈຶ່ງ ໄດ້ອາໄສນັກວິເຄາະລະບົບທີ່ມີຄວາມຮູ້ ແລະ ປະສົບການສູງເນື່ອງຈາກວ່າຫາກນັກວິເຄາະລະບົບບໍ່ ເຂົ້າໃຈເຖິງບັນຫາອັນແທ້ຈິງທີ່ເກີດຂຶ້ນ ກໍຈະບໍ່ສາມາດພັດທະນາລະບົບຂຶ້ນມາເພື່ອແກ້ໄຂບັນຫາໃຫ້ຖືກ ຈຸດໄດ້ ແລະ ມັກຈະມີໂຄງການພັດທະນາລະບົບຫຼາຍໂຄງການທີ່ຫຼັງຈາກໄດ້ດຳເນີນການພັດທະນາ ແລະ ນຳມາໃຊ້ງານແລ້ວປະກົດວ່າບໍ່ສາມາດຕອບສະໜອງຄວາມຕ້ອງການຂອງຜູ້ໃຊ້ງານເຊິ່ງຖືວ່າ ເປັນເລື່ອງທີ່ກໍ່ໃຫ້ເກີດຄວາມສູນເສຍທັງທາງດ້ານການລົງທຶນ ແລະ ໄລຍະເວລາ.

ສະຫຼຸບລວມແລ້ວໄລຍະຂອງການວາງແຜນໂຄງການປະກອບມີກິດຈະກຳຕ່າງໆດັ່ງນີ້:

- ກຳນົດບັນຫາ Problem Definition
- ສຶກສາຄວາມເປັນໄປໄດ້ຂອງໂຄງການ Feasibility study
- ສ້າງຕາຕະລາງກຳນົດເວລາໂຄງການ Project scheduling
- ຈັດຕັ້ງທີມງານໂຄງການ Staff the project

#### **ຂ. ໄລຍະທີ 2 ການວິເຄາະ (Analysis Phase)**

ໄລຍະການວິເຄາະຈະຕ້ອງມີຄຳຕອບກ່ຽວກັບຄຳຖາມວ່າໃຜເປັນຜູ້ທີ່ໃຊ້ລະບົບ ແລະ ມີຫຍັງແດ່ທີ່ຈະຕ້ອງເຮັດໃນໄລຍະນີ້ນັກວິເຄາະລະບົບຈະຕ້ອງດຳເນີນການໃນຂັ້ນຕອນຂອງການວິເຄາະລະບົບງານປັດຈຸບັນ (Current system) ເພື່ອນຳມາພັດທະນາແນວຄວາມຄິດສຳລັບລະບົບໃໝ່ (New System).

ຈຸດປະສົງຫຼັກຂອງການວິເຄາະຄືຈະຕ້ອງສຶກສາ ແລະ ສ້າງຄວາມເຂົ້າໃຈໃນຄວາມຕ້ອງການຕ່າງໆທີ່ໄດ້ລວບລວມມາ ດັ່ງນັ້ນການລວບລວມຄວາມຕ້ອງການ (Requirements Gathering) ຈຶ່ງຈັດເປັນງານສ່ວນພື້ນຖານຂອງການວິເຄາະລະບົບໂດຍຂໍ້ມູນຄວາມຕ້ອງການເຫຼົ່ານີ້ນັກວິເຄາະລະບົບຈະນຳມາວິເຄາະເພື່ອທີ່ຈະປະເມີນວ່າຄວນມີຫຍັງແນ່ທີ່ລະບົບໃໝ່ຕ້ອງດຳເນີນການ ແລະ ດ້ວຍເຫດນີ້ເອງການກຳນົດລາຍລະອຽດກ່ຽວກັບຄວາມຕ້ອງການຂອງຜູ້ໃຊ້ (User Requirements) ຈະທະວີຄວາມສຳຄັນຫຼາຍຂຶ້ນເປັນລຳດັບສຳລັບລະບົບທີ່ມີຄວາມສັບຊ້ອນສູງ ແລະ ນັກວິເຄາະຕ້ອງເອົາໃນໃສ່ກັບການລວບລວມຄວາມຕ້ອງການຈາກຜູ້ໃຊ້ບໍ່ຄວນກຳນົດຄວາມຕ້ອງການຂຶ້ນເອງໂດຍໃຊ້ຄວາມຄິດສ່ວນຕົວຂອງຕົນເອງເປັນຫຼັກ ຫຼື ປະເມີນຄວາມຕ້ອງການຂອງຜູ້ໃຊ້ລະບົບບໍ່ເຂົ້າກັບຈຸດປະສົງ ແລະ ທາກມີການພັດທະນາລະບົບຕໍ່ໄປຈົນແລ້ວລະບົບງານທີ່ໄດ້ກໍ່ຈະບໍ່ກົງກັບຄວາມຕ້ອງການຂອງຜູ້ໃຊ້ລະບົບຢ່າງແທ້ຈິງເຮັດໃຫ້ຕ້ອງມີຄວາມປັບປຸງ ຫຼື ປ່ຽນແປງພາຍຫຼັງ.

ນັກວິເຄາະລະບົບສາມາດລວບລວມຄວາມຕ້ອງການຕ່າງໆໄດ້ຈາກການສັງເກດ, ການເຮັດວຽກຂອງຜູ້ໃຊ້, ການໃຊ້ເຕັກນິກ, ການສຳພາດ ຫຼື ການໃຊ້ແບບສອບຖາມການອ່ານເອກະສານກ່ຽວກັບການປະຕິບັດງານຂອງລະບົບງານປັດຈຸບັນລະບຽບກົດເກນຂອງບໍລິສັດ ແລະ ການມອບໝາຍຕຳແໜ່ງໜ້າທີ່ຮັບຜິດຊອບເຊິ່ງໃນຊ່ວງຂອງການເກັບກຳຂໍ້ມູນຄວາມຕ້ອງການກໍ່ຈະພົບກັບຜູ້ໃຊ້ໃນລະດັບຕ່າງໆທີ່ເຮັດໃຫ້ຮູ້ເຖິງບັນຫາ ແລະ ແນວທາງການແກ້ໄຂບັນຫາທີ່ແນະນຳໂດຍຜູ້ໃຊ້ ດັ່ງນັ້ນການເກັບກຳຄວາມຕ້ອງການຈຶ່ງເປັນກົດຈະກຳທີ່ສຳຄັນເພື່ອຄົ້ນຫາຄວາມຈິງ ແລະ ຕ້ອງສ້າງຄວາມຕ້ອງການເຊິ່ງກັນ ແລະ ກັນເພື່ອສະຫຼຸບອອກມາເປັນຂໍ້ກຳນົດທີ່ມີຄວາມຊັດເຈນໂດຍຂໍ້ກຳນົດເຫຼົ່ານີ້ເມື່ອຜູ້ໃຊ້ທີ່ກ່ຽວຂ້ອງໄດ້ອ່ານແລ້ວຈະຕ້ອງສຶກສາຄວາມໝາຍໄດ້ກົງກັນ.

ຫຼັງຈາກໄດ້ນຳຄວາມຕ້ອງການຕ່າງໆມາສະຫຼຸບເປັນຂໍ້ກຳນົດທີ່ຊັດເຈນແລ້ວຂັ້ນຕອນຕໍ່ໄປກໍ່ຄືນຳຂໍ້ກຳນົດເຫຼົ່ານັ້ນໄປພັດທະນາອອກມາເປັນຄວາມຕ້ອງການຂອງລະບົບໃໝ່ໂດຍເຕັກນິກທີ່ໃຊ້ກໍ່ຄືການພັດທະນາແບບຈຳລອງຂະບວນການ (Process Model) ເຊິ່ງເປັນແຜນພາບທີ່ໃຊ້ອະທິບາຍເຖິງຂະບວນການທີ່ຕ້ອງດຳເນີນໃນລະບົບວ່າມີຫຍັງແດ່ແລະຕໍ່ໄປກໍ່ຄືດຳເນີນການພັດທະນາແບບຈຳລອງຂໍ້ມູນ (Data Model) ເພື່ອອະທິບາຍເຖິງຂໍ້ມູນທີ່ຈັດເກັບໄວ້ສຳລັບສະໜັບສະໜູນການເຮັກວຽກຕ່າງໆ.

ສະຫຼຸບໄລຍະຂອງການວິເຄາະລະບົບປະກອບມີກົດຈະກຳຕ່າງໆດັ່ງນີ້:

- ວິເຄາະລະບົບງານປັດຈຸບັນ.
- ເກັບກຳຄວາມຕ້ອງການໃນດ້ານຕ່າງໆ ແລະ ນຳມາວິເຄາະເພື່ອສະຫຼຸບເປັນຂໍ້ກຳນົດຊັດເຈນ
- ນຳຂໍ້ກຳນົດມາພັດທະນາອອກມາເປັນຄວາມຕ້ອງການຂອງລະບົບໃໝ່.
- ສ້າງແບບຈຳລອງຂະບວນການຂອງລະບົບໃໝ່ໂດຍການແຕ້ມແຜນພາບກະແສຂໍ້ມູນ (Data Flow Diagram: DFD).



- ສ້າງແບບຈຳລອງຂໍ້ມູນໂດຍການແຕ້ມ Entity Relationship Diagram: ERD.

### ຄ. ໄລຍະທີ 3 ການອອກແບບ (Design Phase)

ໄລຍະການອອກແບບເປັນການພິຈາລະນາວ່າລະບົບລະດຳເນີນການໄປໄດ້ແນວໃດເຊິ່ງກ່ຽວຂ້ອງກັບຍຸດທະວິທີການຂອງການອອກແບບທີ່ວ່າດ້ວຍການຕັດສິນໃຈວ່າຈະພັດທະນາລະບົບໃໝ່ດ້ວຍແນວທາງໃດເຊັ່ນພັດທະນາຂຶ້ນເອງ, ຊື້ໂປຣແກຣມສຳເລັດຮູບ ຫຼື ວ່າຈ້າງບໍລິສັດພັດທະນາລະບົບໃຫ້ເປັນຕົ້ນ. ນອກຈາກນີ້ໄລຍະການອອກແບບຈະກ່ຽວຂ້ອງກັບການອອກແບບທາງດ້ານສະຖາປັດຕະຍະກຳລະບົບທີ່ກ່ຽວຂ້ອງກັບອຸປະກອນຮາດແວ, ຊອບແວ ແລະ ເຄືອຂ່າຍ.

ການອອກແບບລາຍງານ (Out Design) ການອອກແບບໜ້າຈໍເພື່ອປະຕິສຳພັນກັບຜູ້ໃຊ້ (User Interface), ການອອກແບບຜັງງານລະບົບ (System Flowchart), ເຊິ່ງລວມເຖິງລາຍລະອຽດຂອງໂປຣແກຣມ (Specific Program), ຖານຂໍ້ມູນ (Database) ແລະ ຂໍ້ມູນທີ່ກ່ຽວຂ້ອງແນວໃດກໍ່ຕາມເຖິງວ່າກິດຈະກຳບາງສ່ວນຂອງໄລຍະອອກແບບນີ້ສ່ວນໃຫຍ່ຈະຖືກດຳເນີນການໄປບາງສ່ວນແລ້ວໃນໄລຍະຂອງການວິເຄາະແຕ່ໄລຍະການອອກແບບນີ້ຈະເນັ້ນເຖິງການດຳເນີນການແກ້ໄຂບັນຫາແນວໃດຫຼາຍກ່ວາໂດຍການນຳຜົນຂອງແບບຈຳລອງທາງ Logical Model. ທີ່ໄດ້ຈາກການວິເຄາະມາພັດທະນາມາເປັນແບບຈຳລອງທາງ Physical Model.

- ການວິເຄາະຈະເນັ້ນແກ້ໄຂບັນຫາຫຍັງແດ່.
- ການອອກແບບຈະເນັ້ນການແກ້ໄຂບັນຫາແນວໃດ.
- ສະຫຼຸບໄລຍະການອອກແບບ.
- ພິຈາລະນາແນວທາງໃນການພັດທະນາລະບົບ.
- ອອກແບບສະຖາປັດຕະຍະກຳລະບົບ (Architecture Design).
- ອອກແບບຖານຂໍ້ມູນ (Database Design).
- ອອກແບບການສະແດງຜົນ (Output Design).
- ອອກແບບການປ້ອນຂໍ້ມູນ (Input Design).
- ອອກແບບສ່ວນຕິດຕໍ່ກັບຜູ້ໃຊ້ (User Interface).
- ສ້າງຕົ້ນແບບ (Prototype).
- ອອກແບບໂປຣແກຣມ (Structure Chart).

### ງ. ໄລຍະທີ 4 ການນຳໄປໃຊ້ (Implementation Phase)

ໃນໄລຍະການນຳໄປໃຊ້ຈະເຮັດໃຫ້ລະບົບເກີດຜົນຂຶ້ນມາໂດຍການສ້າງລະບົບທົດສອບລະບົບ ແລະ ການຕິດຕັ້ງລະບົບໂດຍຈຸດປະສົງຫຼັກຂອງກິດຈະກຳໃນໄລຍະນີ້ບໍ່ແມ່ນພຽງຄວາມໜ້າເຊື່ອຖືຂອງລະບົບ ຫຼື ລະບົບສາມາດເຮັດວຽກໄດ້ດີເທົ່ານັ້ນແຕ່ຕ້ອງໝັ້ນໃຈວ່າຜູ້ໃຊ້ລະບົບຕ້ອງໄດ້ຮັບການເຝິກອົບຮົມເພື່ອໃຊ້ງານລະບົບ ແລະ ຄວາມຄາດຫວັງໃນອົງກອນທີ່ຕ້ອງການຜົນຕອບແທນໃນດ້ານດີກັບການໃຊ້ລະບົບໃໝ່ລຳດັບກິດຈະກຳຕ່າງໆທຸກກິດຈະກຳຕ້ອງເຂົ້າມາດຳເນີນການຮ່ວມກັນໃນໄລຍະນີ້ເພື່ອໃຫ້ລະບົບການປະຕິບັດງານໄດ້ຮັບຄວາມປະສົບຜົນສຳເລັດໄດ້ໂດຍດີ.

ສະຫຼຸບໄລຍະການນຳໄປໃຊ້ຈະປະກອບມີກິດຈະກຳຕ່າງໆດັ່ງຕໍ່ໄປນີ້:

- ສ້າງລະບົບຂຶ້ນມາດ້ວຍການຂຽນໂປຣແກມ.
- ກວດສອບຄວາມຖືກຕ້ອງທາງດ້ານ Verification ແລະ Validation ແລະ ດຳເນີນການທົດສອບລະບົບ.
- ແປງຂໍ້ມູນ (Convert Data).
- ຕິດຕັ້ງລະບົບ (System Installation) ແລະ ສ້າງເອກກະສານຄູ່ມື.
- ຝຶກອົບຮົມຜູ້ໃຊ້ ແລະ ປະເມີນຜົນລະບົບໃໝ່.

ສໍາລັບການສ້າງລະບົບ ຫຼື ການຂຽນໂປຣແກມນັ້ນສາມາດໃຊ້ວິທີການຂຽນໂປຣແກມດ້ວຍພາສາຄອມພິວເຕີເຊັ່ນ: ການໃຊ້ພາສາ Visual Basic, C#, PHP, Java... ນອກຈາກນີ້ຍັງມີເຕັກນິກອື່ນໆເຊັ່ນ: ເຄື່ອງມືໃນການພັດທະນາ Application ເຊິ່ງເປັນຊອບແວທີ່ເປັນແຫຼ່ງລວມຂອງເຄື່ອງມືຕ່າງໆທີ່ໃຊ້ເພື່ອພັດທະນາ Application ເຮັດໃຫ້ຜູ້ຂຽນໂປຣແກມບໍ່ເຮັດວຽກໜັກຄືເມື່ອກ່ອນມີແຕ່ຮຽນຮູ້ແລະປະຍຸກໃຊ້ເຄື່ອງມືເຫຼົ່ານັ້ນກໍສາມາດພັດທະນາລະບົບງ່າຍຂຶ້ນ.

#### ຈ. ໄລຍະທີ 5 ການບໍາລຸງຮັກສາ (Maintenance Phase)

ໂດຍປົກກະຕິແລ້ວໄລຍະການບໍາລຸງຮັກສາຈະບໍ່ນຳເຂົ້າໄປລວມໃນສ່ວນຂອງ SDLC ຈົນກະທັ້ງລະບົບມີການຕິດຕັ້ງເພື່ອໃຊ້ງານແລ້ວເທົ່ານັ້ນໄລຍະນີ້ຈະໃຊ້ເວລາຍາວນານທີ່ສຸດເມື່ອທຽບກັບໄລຍະອື່ນໆທີ່ຜ່ານມາເນື່ອງຈາກລະບົບຈະຕ້ອງໄດ້ຮັບການບໍາລຸງຮັກສາຕະຫຼອດໄລຍະເວລາທີ່ມີການໃຊ້ລະບົບສິ່ງທີ່ຄາດຫວັງຂອງໜ່ວຍງານກໍຄືຕ້ອງການໃຫ້ລະບົບໃຊ້ງານຍາວນານຫຼາຍປີລະບົບສາມາດຮອງຮັບເຕັກໂນໂລຊີໃໝ່ໆໃນອະນາຄົດໄດ້ ດັ່ງນັ້ນໃນຊ່ວງໄລຍະເວລາດັ່ງກ່າວຈຶ່ງສາມາດເພີ່ມເຕີມຄວາມສາມາດຂອງລະບົບໃຫ້ມີປະສິດທິພາບສູງຂຶ້ນພ້ອມທັງການແກ້ໄຂປັບປຸງໂປຣແກມໃນກໍລະນີທີ່ເຫັນຂໍ້ຜິດພາດ.

- ການບໍາລຸງຮັກສາລະບົບ (System Maintenance).
- ການເພີ່ມເຕີມຄວາມສາມາດໃໝ່ໆເຂົ້າໃນລະບົບ (Enhance System).
- ສະໜັບສະໜູນງານຂອງຜູ້ໃຊ້ (Support the User).

ຈາກໄລຍະຕ່າງໆຕາມຂັ້ນຕອນການພັດທະນາລະບົບຕາມແບບແຜນຂອງ SDLC ຈະເຫັນວ່າມີການໃຊ້ຄຳວ່າໄລຍະແລະກິດຈະກຳເຊິ່ງສາມາດອະທິບາຍລາຍລະອຽດເພື່ອໃຫ້ເກີດຄວາມເຂົ້າໃຈກົງກັນດັ່ງນີ້:

- ໄລຍະ (Phase) ຄືກຸ່ມກິດຈະກຳທີ່ກ່ຽວຂ້ອງກັນ.
- ກິດຈະກຳ (Activity) ຄືກຸ່ມຂອງງານທີ່ກ່ຽວຂ້ອງກັນ.
- ໜ້າວຽກ (Task) ຄືງານທີ່ດຳເນີນການເຊິ່ງຖືເປັນງານທີ່ນ້ອຍທີ່ສຸດ.


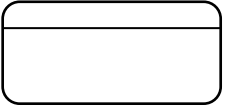
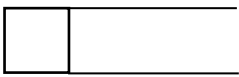
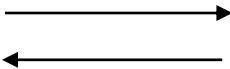

#### 10.1.1.2. ແຜນວາດການໄຫຼຂໍ້ມູນ (Data Flow Diagram DFD)

- ເປັນແຜນພາບທີ່ສະຫຼຸບລວມຂໍ້ມູນທັງໝົດໄດ້ຈາກການວິເຄາະໃນລັກສະນະຂອງຮູບແບບທີ່ເປັນໂຄງສ້າງ.
- ເປັນຂໍ້ຕົກລົງຮ່ວມກັນລະຫວ່າງນັກວິເຄາະລະບົບ ແລະ ຜູ້ຊົມໃຊ້.
- ເປັນແຜນພາບທີ່ໃຊ້ໃນການພັດທະນາຕໍ່ໃນຂັ້ນຕອນຂອງການອອກແບບ.

- ຮູ້ທີ່ໄປທີ່ມາຂອງຂໍ້ມູນທີ່ໄຫຼໃນຂະບວນການຕ່າງໆ.

#### 10.1.1.3. ສັນຍາລັກທີ່ໃຊ້ໃນແຜນວາດຂໍ້ມູນ

ຕາຕະລາງ ສະແດງສັນຍາລັກ Data Flow Diagram

ຊື່	ສັນຍາລັກ	ຄວາມໝາຍ
Boundary Or External Entity		ຂອບເຂດໝາຍເຖິງພາກສ່ວນທີ່ກ່ຽວຂ້ອງເກັບລະບົບ ເຊິ່ງລະບົບບໍ່ສາມາດຄວບຄຸມໄດ້
Process		ປະມວນຜົນຫຼືຫນ້າວຽກທີ່ເຮັດໃນໂຄງການນັ້ນໆ
Data Store		ບ່ອນຈັດເກັບຂໍ້ມູນ
Data Flow		ການໄຫຼຂອງຂໍ້ມູນ
Real-Time Link		ການເຊື່ອມໂຍງແບບໄກທີ່ມີການຕອບກັບແບບທັນທີ

#### 10.1.1.4. ຫຼັກການຂຽນແຜນວາດການໄຫຼຂໍ້ມູນ

##### 1) Process:

- ເມື່ອມີຂໍ້ມູນເຂົ້າໄປທີ່ Process ກໍ່ຕ້ອງມີຂໍ້ມູນ ຫຼື ຜົນຮັບອອກມາຈາກ Process ເຊັ່ນກັນຈະເປັນໄປບໍ່ໄດ້ທີ່ມີສະເພາະຂໍ້ມູນເຂົ້າຢ່າງດຽວ.

##### 2) Data store:

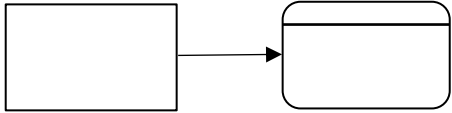
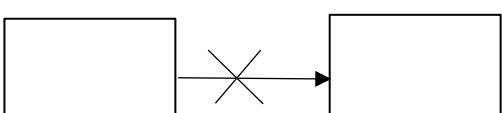
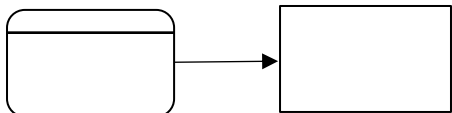
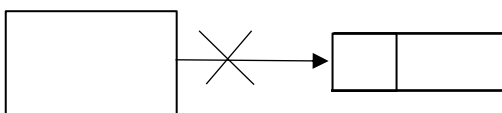
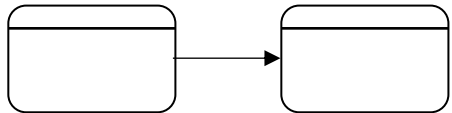
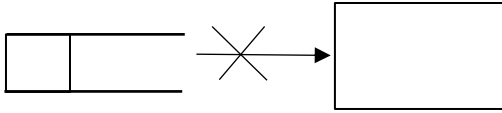
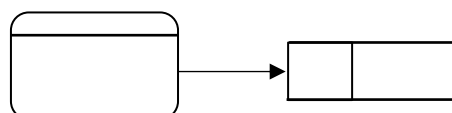
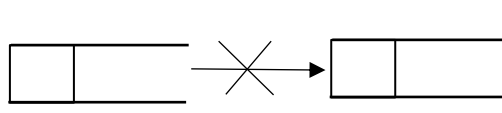
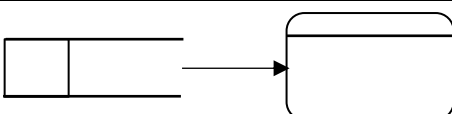
- ຂໍ້ມູນຈະໄຫຼຈາກ Data store ໜຶ່ງໄປຫາ Data store ໜຶ່ງໂດຍກົງບໍ່ໄດ້ຈະຕ້ອງຜ່ານ Process ເທົ່ານັ້ນ
- ຂໍ້ມູນທີ່ສົ່ງຜ່ານ External entity ບໍ່ສາມາດໄຫຼເຂົ້າໄປ Data store ໂດຍກົງໄດ້ຈະຕ້ອງໃຊ້ Process ເປັນຕົວກາງໃນການເຊື່ອມໂຍງເພື່ອຈັດເກັບຂໍ້ມູນໃນ Data store
- ຂໍ້ມູນທີ່ໄຫຼຜ່ານຈາກ Data store ບໍ່ສາມາດເຊື່ອມໂຍງເຂົ້າກັບ External entity ໄດ້ໂດຍກົງຈະຕ້ອງຜ່ານ Process ເທົ່ານັ້ນ.

##### 3) External entity:

- External entity ບໍ່ສາມາດເຊື່ອມໂຍງເຂົ້າຫາກັນໄດ້ຈະຕ້ອງໃຊ້ Process ເປັນຕົວກາງເພື່ອສົ່ງຜ່ານ ແລະ ຊື່ຂອງ External entity ຈະໃຊ້ຄຳນາມເທົ່ານັ້ນ.

4) Data flow:

- ການໄຫຼຂໍ້ມູນທີ່ມີທິວຊີ້ໄປທີ່ Process ໝາຍເຖິງ Process ມີການອ່ານ ຫຼື ການດຶງຂໍ້ມູນຈາກ Data store ມາໃຊ້ວຽກ
- ການໄຫຼຂໍ້ມູນຈາກ Process ທີ່ມີທິວລູກສອນຊີ້ໄປຍັງ Data store ໝາຍເຖິງການ Update ຫຼື ການເພີ່ມຂໍ້ມູນລົງໄປທີ່ Data store

ອະນຸຍາດ	ບໍ່ອະນຸຍາດ
	
	
	
	
	

#### 10.1.1.5. Flowchart


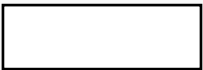
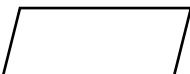
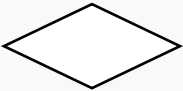
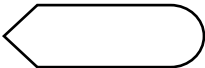


ສັນຍະລັກ Flowchart ຄື ຮູບພາບທີ່ໃຊ້ແທນຄວາມໝາຍການເຮັດວຽກງານໃນລັກສະນະຕ່າງໆ ພາຍໃນແຜນຜັງ (Flowchart) ປະກອບໄປດ້ວຍ ການເລີ່ມຕົ້ນ (Start), ການຈົບ (End), ການກະທຳ (Process), ການນຳເຂົ້າຂໍ້ມູນ (Input), ການສະແດງຜົນຂໍ້ມູນ (Output), ການຕັດສິນໃຈ (Decision), ຄຳອະທິບາຍ (Annotation), ຈຸດເຊື່ອມຕໍ່ (Connector), ທິດທາງການເຮັດວຽກງານ (Direction Flow)

ສັນຍະລັກເຫຼົ່ານີ້ເມື່ອຖືກນຳມາເຊື່ອມຕໍ່ກັນ ຈະກາຍເປັນ "ແຜນຜັງ (Flowchart)" ທີ່ສະແດງລຳດັບຂັ້ນຕອນການເຮັດວຽກງານເພື່ອ

- ເປັນເຄື່ອງມືໃນການຈັດລຳດັບຄວາມຄິດ
- ເຫັນລຳດັບຂັ້ນຕອນການເຮັດວຽກງານທີ່ຊັດເຈນ

(ຄວາມໝາຍສັນຍະລັກ Flowchart ຄວາມໝາຍ ແລະ ວິທີໃຊ້ງານ  
<http://share.olanlab.com/th/it/blog/view/211> Dec 01, 2017).

### ສັນຍະລັກ Flowchart

ຮູບພາບສັນຍະລັກ	ຄວາມໝາຍຂອງສັນຍະລັກ
	ການເລີ່ມຕົ້ນຫລືຈົບ Flowchart (Start ຫລື End)
	ການກະທຳ (Process) ຖືກໃຊ້ເພື່ອສະແດງທີ່ການກະທຳໃນ Flowchart
	ຮັບຂໍ້ມູນ
	ການຕັດສິນໃຈ (Decision)
	ສະແດງຜົນທາງຈຳພາບ
	ຈຸດເຊື່ອມຕໍ່ (Connector)
	ທິດທາງການເຮັດວຽກງານ (Direction Flow)

#### 10.1.2. ທິດສະດີກ່ຽວກັບຖານຂໍ້ມູນ

Database ຫຼື ຖານຂໍ້ມູນຄືກຸ່ມຂອງຂໍ້ມູນທີ່ຖືກເກັບລວບລວມໄວ້ໂດຍມີຄວາມສຳພັນເຊິ່ງກັນ ແລະ ກັນ ໂດຍບໍ່ໄດ້ບັງຄັບວ່າຂໍ້ມູນທັງໝົດນີ້ຈະຕ້ອງເກັບໄວ້ໃນແຟມຂໍ້ມູນດຽວກັນ ຫຼື ແຍກເກັບ ຫຼາຍໆແຟມຂໍ້ມູນ

ລະບົບຖານຂໍ້ມູນຄືລະບົບທີ່ລວບລວມຂໍ້ມູນຕ່າງໆທີ່ກ່ຽວຂ້ອງກັນເຂົ້າໄວ້ດ້ວຍກັນຢ່າງມີລະບົບ, ມີຄວາມສຳພັນລະຫວ່າງຂໍ້ມູນຕ່າງໆທີ່ຊັດເຈນໃນລະບົບຖານຂໍ້ມູນຈະປະກອບດ້ວຍແຟມຂໍ້ມູນຫຼາຍ ແຟມທີ່ມີຂໍ້ມູນກ່ຽວຂ້ອງກັນ, ສຳພັນກັນເຂົ້າໄວ້ດ້ວຍກັນຢ່າງເປັນລະບົບ ແລະ ເປີດໂອກາດໃຫ້ ຜູ້ໃຊ້ ສາມາດໃຊ້ງານ ແລະ ຮັກສາປ້ອງກັນຂໍ້ມູນເຫຼົ່ານີ້ໄດ້ຢ່າງມີປະສິດທິພາບໂດຍມີຊອບແວທີ່ປຸງປະສານ ເພື່ອນຊື່ກາງລະຫວ່າງຜູ້ໃຊ້ ແລະ ໂປຣແກຣມຕ່າງໆທີ່ກ່ຽວຂ້ອງກັບການໃຊ້ຖານຂໍ້ມູນເອີ້ນວ່າລະບົບ ຈັດການຖານຂໍ້ມູນ ຫຼື DBMS

##### 10.1.2.1. ການເຮັດ Normalization

Normalization ເປັນຫຼັກການໜຶ່ງທີ່ຜູ້ອອກແບບຖານຂໍ້ມູນຈະຕ້ອງນຳມາໃຊ້ໃນການແປງຂໍ້ມູນທີ່ຢູ່ໃນຮູບແບບທີ່ຊ້ຳຊ້ອນໃຫ້ຢູ່ໃນຮູບແບບທີ່ງ່າຍຕໍ່ການນຳໄປໃຊ້ງານ ແລະ ກໍ່ໃຫ້ເກີດບັນຫາໜ້ອຍທີ່ສຸດ.

➤ **ຈຸດປະສົງຂອງການເຮັດການເຮັດ Normalization**

- ຫຼຸດຜ່ອນຄວາມຊ້ຳຊ້ອນຂອງຂໍ້ມູນ ເມື່ອຫຼຸດຄວາມຊ້ຳຊ້ອນຈະເຮັດໃຫ້ຫຼຸດເນື້ອທີ່ໃນການຈັດເກັບຂໍ້ມູນ
- ຫຼຸດບັນຫາຄວາມບໍ່ຖືກຕ້ອງຂອງຂໍ້ມູນ ເມື່ອຂໍ້ມູນບໍ່ເກີດຄວາມຊ້ຳຊ້ອນ ເຮັດໃຫ້ການປັບປຸງຂໍ້ມູນສາມາດເຮັດໄດ້ຈາກແຫຼ່ງຂໍ້ມູນພຽງບ່ອນດຽວ.
- ຫຼຸດຄວາມຜິດພາດທີ່ເກີດຈາກການປັບປຸງຂໍ້ມູນ

**ກ. ຂັ້ນຕອນການເຮັດ Normalization**

- Normalization ລະດັບ 1 ຫຼື ເອີ້ນວ່າ 1NF
- Normalization ລະດັບ 2 ຫຼື ເອີ້ນວ່າ 2NF
- Normalization ລະດັບ 3 ຫຼື ເອີ້ນວ່າ 3NF

**ຂ. First Normal Form (1NF)**

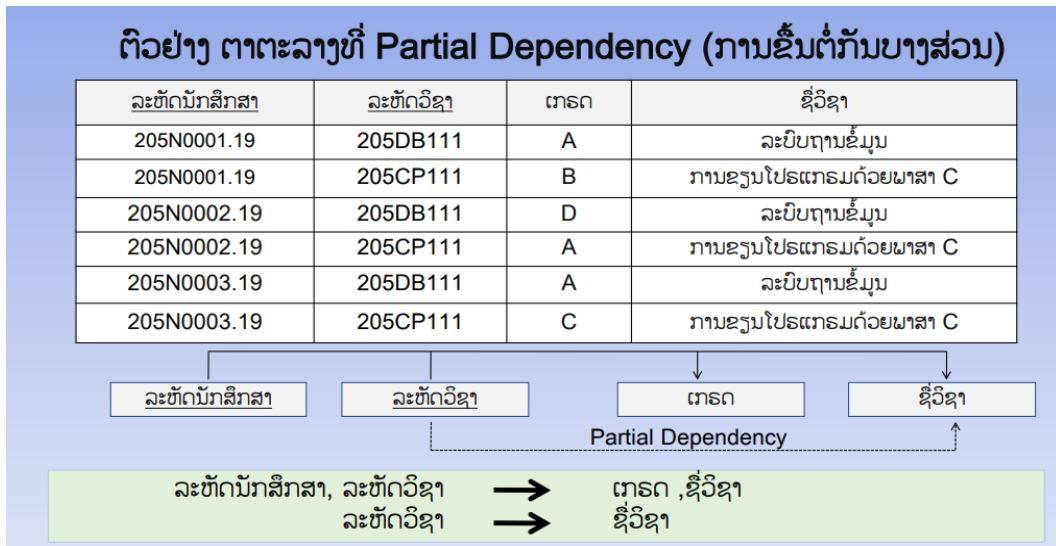
- ທຸກ Attribute ໃນແຕ່ລະ record ຈະເປັນ single value ບໍ່ມີຄ່າຂອງກຸ່ມຂໍ້ມູນຊ້ຳຊ້ອນກັນ (Repeating Group).
- ຂໍ້ມູນທຸກແຖວ (Tuple) ຕ້ອງມີຄ່າບໍ່ຊ້ຳກັນ

ຕາຕະລາງທີ່ມີລັກສະນະຂໍ້ມູນເປັນ Repeating group			
ລະຫັດນັກສຶກສາ	ຊື່	ນາມສະກຸນ	ລະຫັດວິຊາລົງທະບຽນ
205N0001.19	ສົມຊາຍ	ສິງສະນິດ	205DB111 205JV111 205CP111
205Q0001.19	ບຸນມີ	ຄຳປັນຍາ	205JV111 205DB111
ເຮົາສາມາດເຮັດໃຫ້ຢູ່ໃນຮູບ 1NF ຄືດັ່ງນີ້			
ລະຫັດນັກສຶກສາ	ຊື່	ນາມສະກຸນ	ລະຫັດວິຊາລົງທະບຽນ
205N0001.19	ສົມຊາຍ	ສິງສະນິດ	205DB111
205N0001.19	ສົມຊາຍ	ສິງສະນິດ	205JV111
205N0001.19	ສົມຊາຍ	ສິງສະນິດ	205CP111
205Q0001.19	ບຸນມີ	ຄຳປັນຍາ	205JV111
205Q0001.19	ບຸນມີ	ຄຳປັນຍາ	205DB111

**ຄ. Second Normal Form (2NF)**

- ຕ້ອງເປັນ First Normal Form (1NF) ມາກ່ອນ

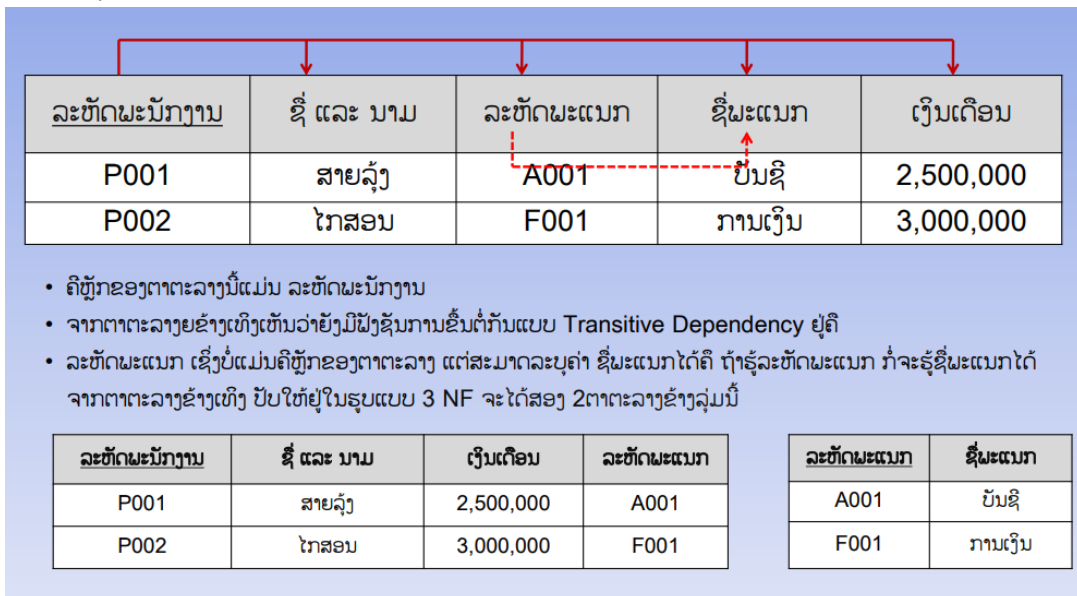
- ຕ້ອງບໍ່ມີ Partial Dependency (ການຂຶ້ນຕໍ່ກັນບາງສ່ວນ)



- ສະຫຼຸບ: ການເຮັດນໍາມາລາຍເຊຊັນ ລະດັບ2 (Second normal form : 2NF) ເປັນການເຮັດໃຫ້ແອດທິບິວທີ່ບໍ່ຂຶ້ນກັບຄືຫຼັກອອກໄປເພື່ອໃຫ້ແອດທິບິວອື່ນທັງໝົດຂຶ້ນກັບສ່ວນທີ່ເປັນຄືຫຼັກ

### ໘. Third Normal Form (3NF)

- Relation ນັ້ນຈະຕ້ອງມີຄຸນສົມບັດ 2NF
- ຕ້ອງບໍ່ມີຄວາມສໍາພັນລະຫວ່າງ Non-key Attribute ຫຼື ບໍ່ມີ Transitive Dependency
- ສະຫຼຸບ: ແອດທິບິວທີ່ບໍ່ແມ່ນຄືຫຼັກ ຕ້ອງບໍ່ຂຶ້ນຕໍ່ກັນເອງ



### ໙. ສະຫຼຸບ Normalization

- 1NF ທຸກແອດທິບິວໃນແຕ່ລະແຖວຕ້ອງມີຂໍ້ມູນພຽງຄ່າດຽວເທົ່ານັ້ນ
- 2NF ຮິເລເຊິນນັ້ນຕ້ອງບໍ່ມີຄວາມສໍາພັນລະຫວ່າງແອດທິບິວແບບບາງສ່ວນ(ແອດທິບິວ ທຸກຕົວຕ້ອງຂຶ້ນກັບຄືຫຼັກທຸກຕົວບໍ່ຂຶ້ນຢູ່ກັບຕົວໃດຕົວໜຶ່ງ)

- 3NF ທຸກແອດທິບິວທີ່ບໍ່ແມ່ນຄືຫຼັກບໍ່ມີຄຸນສົມບັດໃນການກຳນົດຄ່າຂອງແອດທິບິວຕົວອື່ນ

#### 10.1.2.2. ແຜນວາດຄວາມສຳພັນ Entity (ER Diagram)

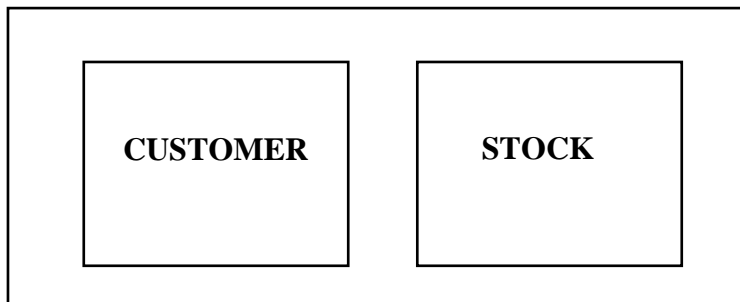
ການອອກແບບຈຳລອງຖານຂໍ້ມູນ ER ນັ້ນກ່ອນອື່ນຕ້ອງຮູ້ຈັກຄວາມໝາຍສັນຍາລັກຕ່າງໆທີ່ໃຊ້ໃນການອອກແບບຈຳລອງ ER ໄດ້ຢ່າງຖືກຕ້ອງ ດັ່ງນັ້ນໃນຫົວຂໍ້ນີ້ຈະເວົ້າເຖິງຄວາມໝາຍ ແລະ ການໃຊ້ງານສັນຍາລັກຕ່າງໆຂອງ ER.

##### 1) ເອັນຕິຕີ (Entity)

ເອັນຕິຕີຄືວັດຖຸທີ່ເຮົາສົນໃຈເຊິ່ງອາດເປັນໄດ້ທັງບຸກຄົນ, ສະຖານທີ່, ວັດຖຸ, ເຫດການ ຫຼື ແນວຄິດທີ່ກໍ່ໃຫ້ເກີດກຸ່ມຂອງຂໍ້ມູນທີ່ຕ້ອງການເອັນຕິຕີແບ່ງອອກເປັນ 2 ປະເພດຄື:

##### 2) Strong Entity:

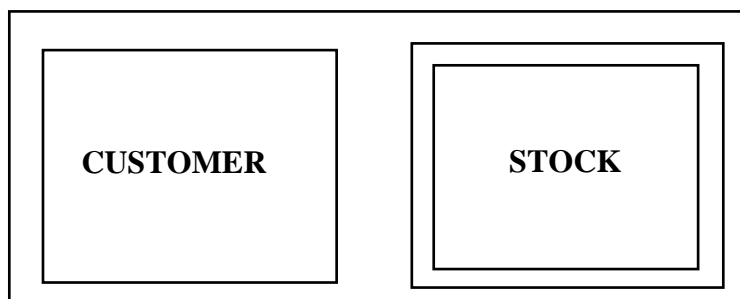
ເປັນເອັນຕິຕີທີ່ເກີດຂຶ້ນດ້ວຍຕົນເອງເປັນອິດສະຫຼະບໍ່ຂຶ້ນກັບເອັນຕິຕີໃດສັນຍາລັກທີ່ໃຊ້ຄືຮູບສີ່ຫຼ່ຽມ ແລະ ສາມາດເອີ້ນ Strong Entity ໄດ້ອີກຊື່ໜຶ່ງວ່າ Regular Entity.



ຮູບທີ 1 ຮູບ Strong Entity

##### 3) Weak Entity:

ເອັນຕິຕີຊະນິດນີ້ຈະຂຶ້ນກັບເອັນຕິຕີຊະນິດອື່ນໆບໍ່ສາມາດເກີດຂຶ້ນໄດ້ຕາມລຳພັງ ແລະ ຈະຖືກລົບເມື່ອເອັນຕິຕີຫຼັກຖືກລົບອອກ ສັນຍາລັກທີ່ໃຊ້ຄືຮູບສີ່ຫຼ່ຽມຊ້ອນກັນ.

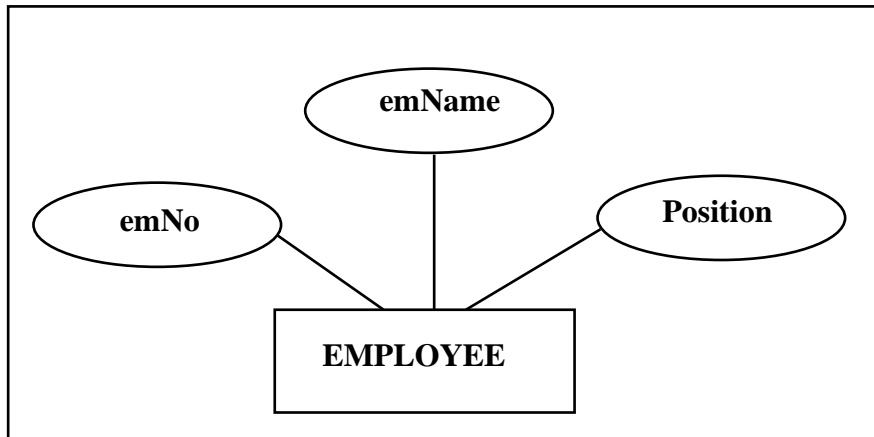


ຮູບທີ 2 ຮູບ Weak Entity

##### 4) ແອັດທິບິວ (Attribute)

ແອັດທິບິວຄືຄຸນສົມບັດຂອງສົມບັດຂອງເອັນຕິຕີສັນຍາລັກຂອງແອັດທິບິວຈະເປັນຮູບວົງມົນ ແອັດທິບິວໃດທີ່ຖືກໃຊ້ເປັນຄືຫຼັກຈະຖືກຂີດເສັ້ນກ້ອງກຳກັບໄວ້.

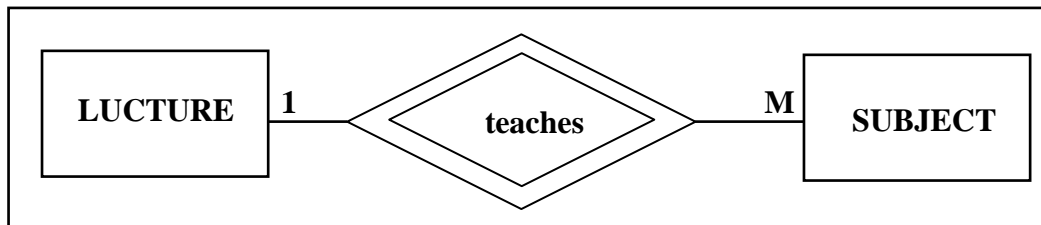




ຮູບທີ 3 ຮູບ Attribute

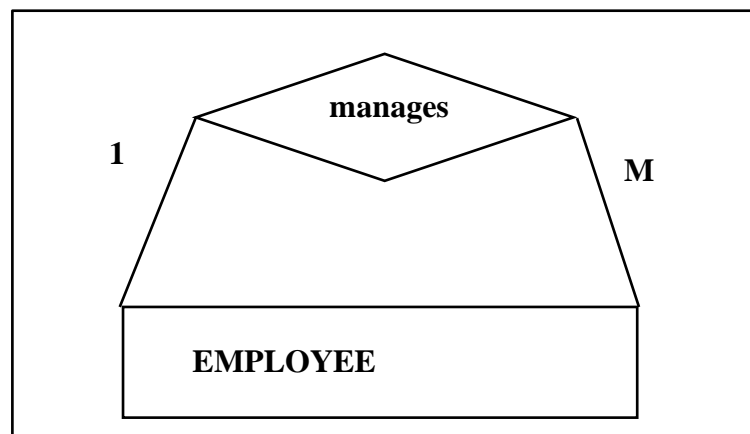
#### 5) ຄວາມສຳພັນ (Relation)

ຄວາມສຳພັນໃນທີ່ນີ້ໝາຍເຖິງຄວາມສຳພັນລະຫວ່າງເອັນຕິຕີໂດຍແຕ່ລະຄວາມສຳພັນຄວນມີຊື່ລະບຸໄວ້ເພື່ອໃຊ້ອະທິບາຍເຊິ່ງປົກກະຕິຈະໃຊ້ສັນຍາລັກຮູບດອກຈັນທີ່ພາຍໃນລະບຸຄຳກິລິຍາໄວ້ເພື່ອອະທິບາຍຄວາມສຳພັນ.



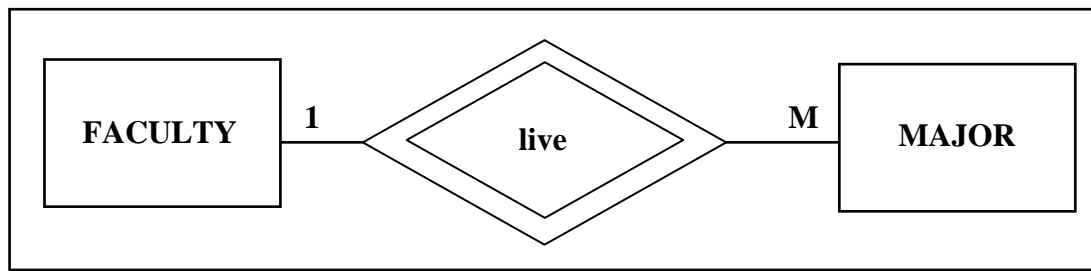
ຮູບທີ 4 ຮູບAttribute

1) ຄວາມສຳພັນແບບຢູນາຣີ (Unary Relationships): ເປັນຄວາມສຳພັນລະຫວ່າງເອັນຕິຕີດຽວໂດຍຄວາມສຳພັນແບບຢູນາຣີນີ້ກໍ່ຄືຄວາມສຳພັນແບບຮີເຄີຊີບ (Recursive) ທີ່ເອີ້ນໃຊ້ໃນຕົວນັ້ນເອງ.



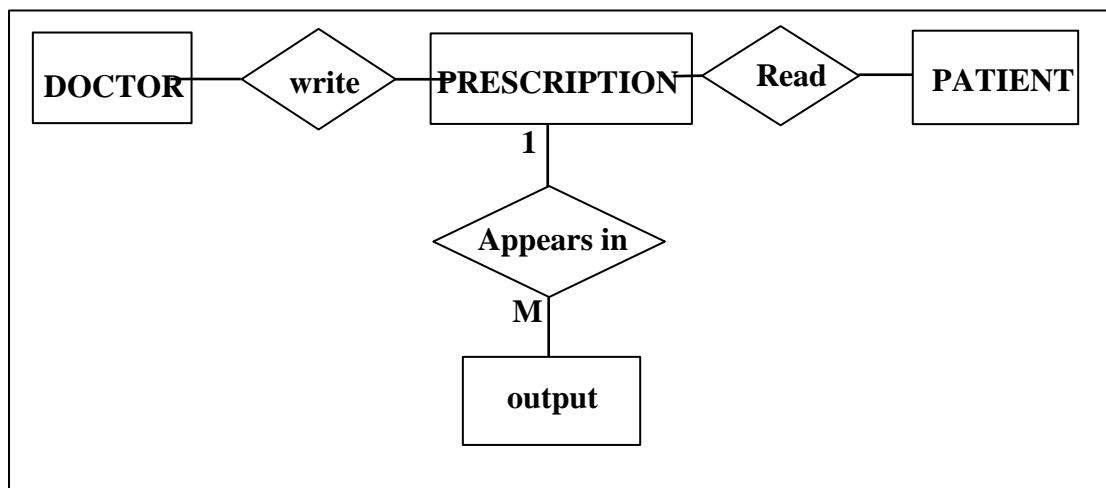
ຮູບທີ 5 ຮູບ Attribute

- 2) ຄວາມສຳພັນແບບໄບນາຣີ (Binary Relationships): ການສຳພັນຊະນິດນີ້ຈະມີເອັນຕິຕີທີ່ກ່ຽວຂ້ອງກັນ 2 ເອັນຕິຕີເຊິ່ງເປັນຄວາມສຳພັນຊະນິດໜຶ່ງທີ່ສາມາດພົບເຫັນໄດ້ຫຼາຍໃນຄວາມເປັນຈິງ.



ຮູບທີ 6 ຄວາມສຳພັນແບບໄບນາຣີ ( Binary Relationships )

- 3) ຄວາມສຳພັນແບບເທີນາຣີ (Ternary Relationships) ເປັນຄວາມສຳພັນທີ່ກ່ຽວຂ້ອງ 3 ເອັນຕິຕີດ້ວຍກັນ.



ຮູບທີ 7 ຄວາມສຳພັນແບບເທີນາຣີ (Ternary Relationships)

### 10.1.2.3. ພາສາ NoSql (Not Only SQL)

ຫຼາຍຄົນຄົງພໍຈະເຄີຍໄດ້ຍິນກັນມາແດ່ກ່ຽວກັບເທັກໂນໂລຢີການຈັດການຂໍ້ມູນແບບໃໝ່ນີ້ ຊຶ່ງກໍຄື NoSQL ເມື່ອເວົ້າເຖິງ NoSQL ຈະໄດ້ຍິນຊື່ເວັບໄຊທີ່ໃຫຍ່ໆ ເຊັ່ນ Facebook, Twitter, FourSquare, Digg ແລະ ອື່ນໆ. ເຮັດໃຫ້ເຮົາຮັບຮູ້ວ່າ NoSQL ເປັນລະບົບຖານຂໍ້ມູນສຳລັບງານທີ່ຕ້ອງຮອງຮັບຂໍ້ມູນຂະໜາດໃຫຍ່ໆຮອງຮັບການຂະຫຍາຍລະບົບໄດ້ງ່າຍເປັນຕົ້ນ.

ຊຶ່ງກໍເປັນເຊັ່ນນັ້ນແທ້ ແຕ່ງານທີ່ນ້ອຍໆຈະເຮັດຢ່າງໃດໃຊ້ງານໄດ້ບໍ່ ຄຸ້ມຄ່າທີ່ຈະນຳ NoSQL ມາໃຊ້ງານຂະໜາດນ້ອຍ ຫຼື ບໍ່ ຫຼືໃຊ້ Relational Database ກໍພຽງພໍແລ້ວ ຄຳຕອບຄືຂຶ້ນຢູ່ກັບລັກສະນະໃນການນຳມາໃຊ້ງານ ກ່ອນທີ່ຈະຕອບຄຳຖາມວ່າ NoSQL ເປັນຄຳຕອບຂອງລະບົບຈັດເກັບຂໍ້ມູນຂອງ ຫຼື ບໍ່ ລອງພິຈາລະນາຫົວຂໍ້ຕ່າງໆດັ່ງຕໍ່ໄປນີ້:

### 1) ຜູ້ໃຊ້ນັບມື້ນັບຫຼາຍ (BigUsers)

ຈະເຫັນໄດ້ວ່າໃນຊ່ວງເວລາທີ່ຜ່ານມານີ້ ແລະ ໃນປັດຈຸບັນຜູ້ທີ່ໃຊ້ງານ Internet ມີແນວໂນ້ມຫລາຍຂຶ້ນເລື້ອຍໆ ບໍ່ວ່າຈະໃຊ້ງານຜ່ານ Desktop PC ຫຼື Smartphone ຊຶ່ງເທັກໂນໂລຊີຂອງອຸປະກອນ (Devices) ມີຫຼາຍຂຶ້ນ ແລະ ໃຊ້ງານໄດ້ງ່າຍຂຶ້ນ

ການພັດທະນາລະບົບໃຫ້ສາມາດຮອງຮັບປະລິມານການເຂົ້າໃຊ້ງານແຕ່ລະອຸປະກອນ (Devices) ເປັນສິ່ງໜຶ່ງທີ່ຕ້ອງນຳມາພິຈາລະນາ ແລະ ບໍ່ພຽງແຕ່ຕ້ອງຮອງຮັບການເຂົ້າໃຊ້ງານຂອງຜູ້ໃຊ້ໄດ້ເທົ່ານັ້ນ ເຮົາຕ້ອງຮອງຮັບວິທີການປ້ອນຂໍ້ມູນແບບໃໝ່ຄືແຕ່ກ່ອນຜູ້ຈັດການເນື້ອຫາຕ່າງໆ ຄື Web Master, Web Editor, ຜູ້ເບິ່ງແຍງລະບົບເປັນຕົ້ນ ແຕ່ປະຈຸບັນຜູ້ທີ່ປ້ອນຂໍ້ມູນຄືຜູ້ໃຊ້ບໍລິການ (users) ໂດຍກົງຜ່ານອຸປະກອນ (Devices) ຕ່າງໆທີ່ມີຫຼາກຫຼາຍ ແລະ ການປ້ອນຂໍ້ມູນກໍ່ງ່າຍກວ່າແຕ່ກ່ອນອີກດ້ວຍ

ຍັງມີປັດໄຈອື່ນໆເຊັ່ນ ເທດສະການສຳຄັນໆທີ່ຄົນຈະເຂົ້າມາໃຊ້ງານຫລາຍເປັນພິເສດ ຫຼື ຜູ້ໃຊ້ງານທີ່ບໍ່ແມ່ນແຕ່ປະເທດເຮົາເທົ່ານັ້ນ ເພາະໂລກອິນເຕີເນັດເຖິງກັນ ອາດຈະຕ້ອງເບິ່ງວ່າລະບົບເຮົາມີຜູ້ເຂົ້າໃຊ້ງານຈາກຕ່າງປະເທດ ຫຼື ທົ່ວໂລກ ຫຼື ບໍ່ອີກດ້ວຍ

ດັ່ງນັ້ນ ເຮົາຕ້ອງກັບມາພິຈາລະນາວິທີການຈັດການຖານຂໍ້ມູນແລ້ວວ່າ ບໍ່ແມ່ນແຕ່ເຮັດໃຫ້ຮອງຮັບກັບການເຂົ້າມາໃຊ້ງານຂອງຜູ້ໃຊ້ບໍລິການເທົ່ານັ້ນ ແຕ່ຕ້ອງຮອງຮັບການຈັດເກັບຂໍ້ມູນທີ່ຫລາຍຂຶ້ນເລື້ອຍໆໄດ້ອີກ

### 2) ປະເພດຂໍ້ມູນຕ່າງໆ ແລະ ຂໍ້ມູນທີ່ຕ້ອງການຈັດເກັບຫຼາຍຂຶ້ນເລື້ອຍໆ (BigData)

ຈາກຕົວແປຂອງຜູ້ໃຊ້ງານມີຫລາຍຂຶ້ນເລື້ອຍໆ ອຸປະກອນໃນການເຂົ້າໃຊ້ງານກໍ່ຫຼາກຫຼາຍປະເພດຂອງຂໍ້ມູນທີ່ໄດ້ຈາກແຕ່ລະອຸປະກອນກໍ່ຫຼາກຫຼາຍປະເພດເຊັ່ນ: ຂໍ້ຄວາມ, ຮູບພາບ, ສຽງ, ວິດີໂອ ຕຳແໜ່ງສະຖານທີ່ (GeoLocation) ແລະ ອື່ນໆ ແລະ ການປ້ອນຂໍ້ມູນເຫຼົ່ານີ້ກໍ່ງ່າຍແສນງ່າຍ ເພາະເທັກໂນໂລຊີຂອງຮາດແວຣ ແລະ ຊອບແວມີການພັດທະນາຫລາຍຂຶ້ນເລື້ອຍໆ ໃຊ້ງານງ່າຍຂຶ້ນສະດວກຂຶ້ນວ່ອງໄວຂຶ້ນເລື້ອຍໆ

ດັ່ງນັ້ນ ການຈັດເກັບຂໍ້ມູນທີ່ລ້ຽງໄຫຼເຂົ້າມາຈາກອຸປະກອນຕ່າງໆເຫຼົ່ານີ້ ເຮົາອາດຈະຕ້ອງນຳມາວິເຄາະພຶດຕິກຳຂອງຜູ້ໃຊ້ບໍລິການ ການສົ່ງເສີມການຕະຫຼາດ ເຮັດຂໍ້ມູນການຕັດສິນໃຈຂອງຜູ້ບໍລິຫານຂໍ້ມູນລູກຄ້າສຳພັນ ແລະ ອື່ນໆອີກຫລວງຫລາຍ ການນຳລະບົບຖານຂໍ້ມູນແບບເດີມ (Relational Database) ອາດຈະບໍ່ເໝາະກັບລັກສະນະງານບາງຢ່າງອີກຕໍ່ໄປ

### 3) ເຕັກໂນໂລຊີຮາດແວໄດ້ປ່ຽນແປງ ລາຄາກໍ່ຖືກຫຼຸດລົງແຕ່ວ່າປະສິດທິພາບດີຂຶ້ນ (Cloud Computing).

ເຮົາອາດຈະເລີ່ມໄດ້ຍິນຄຳວ່າ Cloud Technology ຊຶ່ງກໍ່ມີຫຼາຍປະເພດແຕ່ໃນທີ່ນີ້ເຮົາຈະເວົ້າເຖິງໃນແງ່ຂອງການນຳມາໃຊ້ງານໂດຍປະຈຸບັນຖ້າໃຜເຄີຍໄດ້ລອງໃຊ້ງານ EC2 ຂອງ Amazon ມາແດ່ແລ້ວຈະຮູ້ຈັກເປັນຢ່າງດີວ່າການຈະມີເຄື່ອງ Server ແຮງໆຈັກເຄື່ອງ ເປັນເລື່ອງທີ່ງ່າຍຫລາຍ ຫຼື ການຈະມີ Server 10 ເຄື່ອງ ຫຼື 20 ເຄື່ອງນຳມາຕໍ່ເປັນ Database Cluster ນັ້ນງ່າຍຫຼາຍພຽງແຕ່ຄລິກສ້າງ Instance ບໍ່ຈັກເທື່ອກໍ່ໄດ້ Server ມາໃຊ້ງານແລ້ວ ແລະ ລາຄາກໍ່ຖືກຫລາຍໆຖ້າເຮົາບໍ່ໃຊ້ງານ

ແລ້ວກໍຍົກເລີກການໃຊ້ງານ ແລະ ຄົນກັບໄປໄດ້ທັນທີ ເມື່ອທຽບກັບສະໄໝກ່ອນທີ່ເຮົາຕ້ອງການມີ Server ຈັກ 10 ເຄື່ອງຈະຕ້ອງລົງທຶນຊື້ເຄື່ອງມາຫລາຍ ຖ້າເຊົາໃຊ້ງານແລ້ວຕ້ອງແບກຮັບພາລະເຄື່ອງ Server ເຫຼົ່ານີ້ໄວ້ຊຶ່ງເປັນຕົ້ນທຶນທີ່ແພງຫລາຍ

ຈາກທີ່ກ່າວມາຂ້າງເທິງ ເຮົາບໍ່ໄດ້ເນັ້ນຂໍ້ດີຂອງ EC2 ແຕ່ຢ່າງໃດແຕ່ກໍາລັງຈະເນັ້ນວ່າພາບລວມຂອງການໃຊ້ງານ Server ເລີ່ມປ່ຽນໄປຄືໃຊ້ງານໄດ້ງ່າຍຂຶ້ນ ລາຄາຖືກລົງ ແຕ່ປະສິດທິພາບດີຂຶ້ນ ຊຶ່ງເປັນສິ່ງທີ່ສໍາຄັນໃນການນໍາມາພິຈາລະນາຂອງເທັກໂນໂລຢີດ້ານຖານຂໍ້ມູນຄື ຖ້າຕ້ອງການຈັດເກັບຖານຂໍ້ມູນໃຫຍ່ໆ ຫຼື ຮອງຮັບຜູ້ໃຊ້ງານໄດ້ເປັນຈຳນວນຫລາຍໆ ການຂະຫຍາຍລະບົບຖານຂໍ້ມູນເປັນເລື່ອງທີ່ງ່າຍຂຶ້ນ ຊຶ່ງເຮັດໄດ້ໂດຍການເອົາ Server ມາຕັ້ງກັນອອກໄປ ຫຼື ເອີ້ນວ່າການຂະຫຍາຍອອກແນວນອນ (Scale Out) ບໍ່ແມ່ນການຂະຫຍາຍລະບົບຄືແຕ່ກ່ອນ ຄືຂະຫຍາຍອອກແນວຕັ້ງ (Scale Up) ແລະ ຕ້ອງໃຊ້ເຄື່ອງ Server ທີ່ມີປະສິດທິພາບສູງໆ ຊຶ່ງຈະມີຕົ້ນທຶນທີ່ແພງກວ່າການຂະຫຍາຍແບບແນວນອນຫລາຍ

ດັ່ງນັ້ນ ການຂະຫຍາຍລະບົບທີ່ຢູ່ເທິງພື້ນຖານຂອງ NoSQL ຄືຮອງຮັບການຂະຫຍາຍລະບົບແບບແນວນອນ (Scale Out) ຊຶ່ງຈະກະຈາຍຂໍ້ມູນໄປເກັບທີ່ເຄື່ອງ Server ຫຼາຍໆເຄື່ອງ ແລະ ໃຊ້ເຄື່ອງ Server ທົ່ວໄປທີ່ເອີ້ນວ່າ (Commodity Server) ບໍ່ຈໍາເປັນຕ້ອງໃຊ້ Server ທີ່ເອີ້ນວ່າ Enterprise Server ທີ່ມີລາຄາແພງຕາມ Spec ທີ່ສູງຂຶ້ນເລື້ອຍໆ ແລະ ການບໍລິຫານຈັດການກໍຍາກຂຶ້ນອີກດ້ວຍ

#### 4) ບັນຫາຂອງ Relational Database

ຖ້າໃຜທີ່ໃຊ້ງານ Relation Database ທີ່ຕ້ອງການຮອງຮັບການຈັດເກັບຂໍ້ມູນຂະໜາດໃຫຍ່ໆ ຄົງຫຼີກລ້ຽງບໍ່ໄດ້ເລື່ອງການເຮັດ Sharding ແລະ Distributed Cache ເພາະເປັນຕົວຫຼັກທີ່ຕ້ອງເຮັດເພື່ອຂະຫຍາຍລະບົບຖານຂໍ້ມູນຂອງ Relational Database ໃຫ້ສາມາດຮອງຮັບຂໍ້ມູນທີ່ຫລາຍຂຶ້ນ ແລະ ຮອງຮັບຈຳນວນການເຂົ້າມາໃຊ້ງານລະບົບໄດ້ຫລາຍຂຶ້ນ

#### 5) Manual Sharding

ການແບ່ງຕາຕະລາງຖານຂໍ້ມູນ (Table) ອອກເປັນສ່ວນໆ ແລ້ວກໍການກະຈາຍໄປຈັດເກັບໃນຫຼາຍໆ Server ເພື່ອໃຫ້ແຕ່ລະຕາຕະລາງ (Table) ຂອງຖານຂໍ້ມູນບໍ່ຈັດເກັບຂໍ້ມູນທີ່ຫຼາຍເກີນໄປ ເພາະຖ້າຂໍ້ມູນໃນແຕ່ລະຖານຂໍ້ມູນຫຼາຍເກີນໄປ ຈະເຮັດໃຫ້ລະບົບຖານຂໍ້ມູນຊ້າ ແຕ່ບັນຫາກໍຈະຕາມມາອີກຄື ເມື່ອຕ້ອງກະຈາຍຂໍ້ມູນອອກໄປໃນແຕ່ລະ Server ການຈະເກັບຂໍ້ມູນເຊັ່ນ: ເພີ່ມ, ແກ້ໄຂ, ລຶບ, ດຶງຂໍ້ມູນມາສະແດງຕ່າງໆ ຈະຕ້ອງເຮັດຜ່ານ Application ຫຼື ຕ້ອງມີ Server ບາງໂຕທີ່ຖ້າດຶງຂໍ້ມູນແຕ່ລະ Server ມາທັງໝົດເປັນກ້ອນດຽວ ນັ້ນໝາຍຄວາມວ່າ ເຮົາຕ້ອງເຮັດດ້ວຍໂຕເຮົາເອງ ບໍ່ແມ່ນລະບົບຖານຂໍ້ມູນຈັດການໃຫ້ (Manual Sharding)

#### 6) Distributed Cache

ເມື່ອເຮົາຕ້ອງການໃຫ້ລະບົບຮອງຮັບການເຂົ້າມາໃຊ້ງານຫລາຍໆໄດ້ນັ້ນ ຖ້າຈະຕ້ອງເຂົ້າມາອ່ານຂໍ້ມູນຜ່ານ Database ໂດຍກົງມັນອາດຈະຮອງຮັບບໍ່ໄຫວ ຫຼື ເຮັດໄດ້ຊ້າ ດັ່ງນັ້ນ ຈະຕ້ອງມີການເຮັດ Cache Layer ຂຶ້ນມາ ຄືແທນທີ່ຈະເຂົ້າໄປອ່ານຈາກຖານຂໍ້ມູນໂດຍກົງ ກໍໃຫ້ອ່ານຜ່ານ Cache ກ່ອນ

ດັ່ງນັ້ນ ການອ່ານຂໍ້ມູນຈາກ Cache ເປັນການອ່ານຈາກ Memory ໂດຍກົງເຮັດໃຫ້ຮອງຮັບປະລິມານ ການເຂົ້າມາໃຊ້ງານໄດ້ຫຼາຍຂຶ້ນ

ແຕ່ບັນຫາຄືການເຮັດ Cache Layer ນີ້ຮອງຮັບສະເພາະການອ່ານຂໍ້ມູນເທົ່ານັ້ນ ບໍ່ຮອງຮັບການ ຂຽນຂໍ້ມູນໄດ້ ຖ້າຕ້ອງການຮອງຮັບການຂຽນຂໍ້ມູນປະລິມານຫຼາຍໆ ແລະ ອ່ານຂໍ້ມູນປະລິມານຫຼາຍໆ ຈຶ່ງເປັນສິ່ງທີ່ Relational Database ບໍ່ສາມາດຮອງຮັບງານໃນລັກສະນະ ອ່ານ,ຂຽນ ຂໍ້ມູນປະລິມານ ຫຼາຍໆໄດ້ດີ ແລະ ສິ່ງສໍາຄັນການເຮັດ Cache Layer ຈະຕ້ອງມີການດູແລຮັກສາ ແລະ ໃຊ້ Server ແຍກອອກໄປຕ່າງຫາກອີກ

ຈາກຈຸດນີ້ເອງທັງການເຮັດ Sharding ແລະ Caching ເປັນສິ່ງທີ່ຖືກພັດທະນາຂຶ້ນໃນ NoSQL ເທັກໂນໂລຢີ ໂດຍຮອງຮັບ Auto-Sharding ແລະ Integrated Caching ໃນຕົວເອງ ດັ່ງນັ້ນ ເຮົາຈຶ່ງ ໄດ້ເຫັນ NoSQL ຖືກນຳໄປໃຊ້ງານກັບລະບົບໃຫຍ່ໆເຊັ່ນ Facebook, Twitter, FourSquare, Digg ແລະ ອື່ນໆ. ເພາະວ່າ NoSQL ອອກແບບມາເພື່ອຮອງຮັບຄວາມຕ້ອງການງານໃຫຍ່ໆໄດ້ດີໂດຍ ສະເພາະຢູ່ແລ້ວ ແຕ່ເຖິງຢ່າງໃດກໍຕາມຍັງມີຄຸນສົມບັດອື່ນໆທີ່ໜ້າສົນໃຈໃນ NoSQL ເທັກໂນໂລຢີ

### ກ. ຄຸນສົມບັດຂອງ NoSQL Database

#### - Dynamic Schemas

ການຈັດເກັບຂໍ້ມູນຕ່າງໆໃນຖານຂໍ້ມູນແບບ Relational Database ເຮົາຈະຕ້ອງມີການສ້າງ Schema ຫຼື ຮູບແບບຂອງໂຄງສ້າງຕາຕະລາງວ່າຈະຈັດເກັບຂໍ້ມູນຫຍັງ ເມື່ອຕ້ອງການຈັດເກັບຂໍ້ມູນ ເພີ່ມເຕີມຕ້ອງປ່ຽນ Schema ພາຍຫຼັງ (Alter-Table) ກ່ອນຈະຈັດເກັບຂໍ້ມູນຮູບແບບໃໝ່ໄດ້.

ແຕ່ໃນປະຈຸບັນການຈັດເກັບຂໍ້ມູນມີການປ່ຽນແປງຕະຫລອດເວລາ ເພາະຄວາມຕ້ອງການຈັດເກັບ ຂໍ້ມູນຕ່າງໆມີຫຼາກຫຼາຍຂຶ້ນເລື້ອຍໆ ການກຳນົດໂຄງສ້າງຂອງຕາຕະລາງຖານຂໍ້ມູນ ຫຼື ການຕ້ອງປ່ຽນ ໂຄງສ້າງຖານຂໍ້ມູນເລື້ອຍໆ ໂດຍທີ່ຂໍ້ມູນຍັງມີຢູ່ແລ້ວເປັນເລື່ອງທີ່ຍາກຫຼາຍ ຫຼື ເຮັດບໍ່ໄດ້ເລີຍ ວິທີ ການຄືອາດຈະຕ້ອງແຍກອອກເປັນຕາຕະລາງໃໝ່ຊຶ່ງເປັນວິທີແກ້ບັນຫາຊົ່ວຄາວເທົ່ານັ້ນ.

ລະບົບຖານຂໍ້ມູນແບບ NoSQL ເຮົາບໍ່ຈຳເປັນຕ້ອງມີ Schema ທີ່ຕາຍຕົວ ຫຼື ບໍ່ຕ້ອງມີ Schema ກ່ອນທີ່ຈະຈັດເກັບຂໍ້ມູນ ຂໍ້ມູນແຕ່ລະແຖວສາມາດຈັດເກັບໄດ້ຕາມຕ້ອງການ ຈະເພີ່ມ ຫຼື ຫຼຸດ ກໍບໍ່ມີ ບັນຫາກັບລະບົບ ເຮັດໃຫ້ເຮົາສາມາດຈັດເກັບຂໍ້ມູນໄດ້ຕາມທີ່ຕ້ອງການປ່ຽນແປງໄດ້ຕະຫລອດເວລາ ສະດວກ ແລະ ວ່ອງໄວ.

#### - Auto-Sharding

ເມື່ອຂໍ້ມູນມີຂະໜາດໃຫຍ່ ຫຼື ເຮົາຕ້ອງການເພີ່ມປະສິດທິພາບການອ່ານ ແລະ ຂຽນຂໍ້ມູນປະລິ ມານຫຼາຍໆ ການເຮັດ Sharding ໃນລະບົບ NoSQL Database ຈະກໍານົດການຈະຈາຍຂໍ້ມູນໄປຍັງ Server ຕ່າງໆອັດໂນມັດ (Auto-Sharding) ຜູ້ພັດທະນາ (Developer) ບໍ່ຕ້ອງຂຽນໂປຣແກຣມໃນ ການຈະຈາຍຂໍ້ມູນເອງຄືກັບ Relational Database

ການຈະຈາຍຂໍ້ມູນອອກໄປຫຼາຍໆ Server ນີ້ຍັງເຮັດໃຫ້ມີຂໍ້ດີຄື ປະຢັດຕົ້ນທຶນໃນການຂະຫຍາຍ ລະບົບ ເພາະເປັນການຂະຫຍາຍແບບແນວນອນ (Scale Out) ຊຶ່ງສາມາດນຳ Server ປົກກະຕິທີ່ວ ໄປມາໃຊ້ງານໄດ້ ບໍ່ຈຳເປັນຕ້ອງເປັນ Enterprise Server

## - **Replication**

ການສຳເນົາຂໍ້ມູນຈາກເຄື່ອງໜຶ່ງໄປອີກເຄື່ອງໜຶ່ງ (Replication) ເມື່ອ Server ໜຶ່ງເສຍຫາຍ ອີກເຄື່ອງໜຶ່ງຈະຂຶ້ນມາເຮັດວຽກແທນທັນທີໂດຍຂໍ້ມູນຂອງແຕ່ລະເຄື່ອງຈະມີຂໍ້ມູນຄືກັນ ດັ່ງນັ້ນ Replication ເປັນໜຶ່ງຄຸນສົມບັດທີ່ຕອບສະໜອງຕໍ່ການໃຊ້ງານທີ່ຕ້ອງການຄວາມຕໍ່ເນື່ອງໄດ້ ຕະຫລອດເວລາ (High Availability)

## - **Integrated Caching**

ການຈັດເກັບຂໍ້ມູນທີ່ໃຊ້ງານເລື້ອຍໆ ເຂົ້າໄວ້ໃນ Memory (RAM) ຊຶ່ງເປັນຄຸນສົມບັດເດັ່ນຂອງ NoSQL ທີ່ທັງຫມົດ Caching ໄວ້ໃນຕົວເອງຢູ່ແລ້ວ ເຮົາບໍ່ຈຳເປັນຕ້ອງເຮັດ Cache Layer ຄືກັບ Relational Database ອີກຕໍ່ໄປ ທີ່ຕ້ອງເຮັດ Cache Layer ແຍກຕ່າງຫາກ ແລະ ເບິ່ງແຍງຮັກສາ ແຍກອອກໄປຕ່າງຫາກອີກດ້ວຍ

## **ຂ. ປະເພດຂອງຖານຂໍ້ມູນ NoSQL**

NoSQL ຖືກແບ່ງປະເພດຕາມລັກສະນະການຈັດເກັບຂໍ້ມູນທີ່ແຕກຕ່າງກັນ ດັ່ງນັ້ນ ການຈະເລືອກ NoSQL Database ໂຕໃດໂຕໜຶ່ງຈະຕ້ອງເບິ່ງອີກວ່າການຈັດເກັບຂໍ້ມູນຂອງຖານຂໍ້ມູນເປັນແບບໃດ ເຊັ່ນ

- Document databases ເຊັ່ນ MongoDB, CouchDB, Elasticsearch
- Graph stores ເຊັ່ນ Neo4J, Infinite Graph, InfoGrid
- Key-value stores ເຊັ່ນ DynamoDB, Redis, MemcacheDB
- Wide-column stores ເຊັ່ນ Cassandra, Amazon SimpleDB, Hadoop / HBase

## **ຄ. Open source License**

ໂດຍສ່ວນໃຫຍ່ແລ້ວ NoSQL ຈະເປັນລິຂະສິດແບບ Open source ຊຶ່ງບໍ່ຕ້ອງເສຍຄ່າໃຊ້ຈ່າຍໃນການນຳມາໃຊ້ງານ ດັ່ງນັ້ນ ເຮົາສາມາດນຳ NoSQL Database ແຕ່ລະຕົວມາຕິດຕັ້ງໃຊ້ງານໄດ້ ໂດຍບໍ່ເສຍຄ່າໃຊ້ຈ່າຍໃດໆ (ຟຣີ)

## **ງ. ນຳ NoSQL ມາໃຊ້ງານຂະໜາດນ້ອຍໄດ້ ຫຼື ບໍ່?**

ຈາກທີ່ກ່າວມາແລ້ວ ຄົງພໍຈະຕອບຄຳຖາມນີ້ໄດ້ວ່າການນຳ NoSQL Database ເມື່ອນຳມາໃຊ້ໃນງານຂະໜາດໃຫຍ່ນັ້ນເໝາະສົມຢ່າງແນ່ນອນ ແຕ່ຖ້າເປັນລະບົບທີ່ວຽກໄປຄວນຈະນຳ NoSQL ມາໃຊ້ງານ ຫຼື ບໍ່

ຄຳຕອບຄື ຂຶ້ນຢູ່ກັບລັກສະນະວຽກວ່າເຮົາຈະໃຊ້ຄຸນສົມບັດຫຍັງຂອງ NoSQL ຖ້າເຮົາຕ້ອງການຈັດເກັບຂໍ້ມູນທີ່ບໍ່ຕ້ອງຢຶດຕິດກັບໂຄງສ້າງ (Dynamic Schema) ແລະ ຕ້ອງການເຂົ້າໃຊ້ງານລະບົບທີ່ວ່ອງໄວ (Integrated Caching) ຂໍ້ມູນອາດຈະຍັງບໍ່ຫຼາຍເທົ່າໃດອາດຈະໃຊ້ NoSQL ໄດ້ຢ່າງແນ່ນອນ ແຕ່ຖ້າບໍ່ຕ້ອງການໃຊ້ງານ (Dynamic Schema) ບໍ່ຕ້ອງການເຂົ້າໃຊ້ງານທີ່ວ່ອງໄວ (Integrated Caching) ເພາະໃຊ້ Relational Database ກໍເຮັດໄດ້ດີຢູ່ແລ້ວ Database Server ກັບ Web Server ກໍຢູ່ທີ່ Server ດຽວກັນ ຂໍ້ມູນບໍ່ຫລາຍນັ້ນ ບໍ່ຕ້ອງການຈັດເກັບຂໍ້ມູນທີ່ເພີ່ມຂະຫຍາຍຂຶ້ນທຸກມື້ໆ ຜູ້ເຂົ້າ

ໃຊ້ງານກໍບໍ່ໄດ້ຫລາຍ ເບິ່ງແລ້ວວ່າລະບົບບໍ່ມີແນວໂນ້ມຈະຕ້ອງຂະຫຍາຍລະບົບໃນອະນາຄົດອັນໃກ້  
ສາມາດໃຊ້ງານ Relational Database ໄດ້ດີຢູ່ແລ້ວຢ່າງບໍ່ມີບັນຫາ.

#### 10.1.2.4. ລະບົບຈັດການຖານຂໍ້ມູນ (Database Management Systems: DBMS)

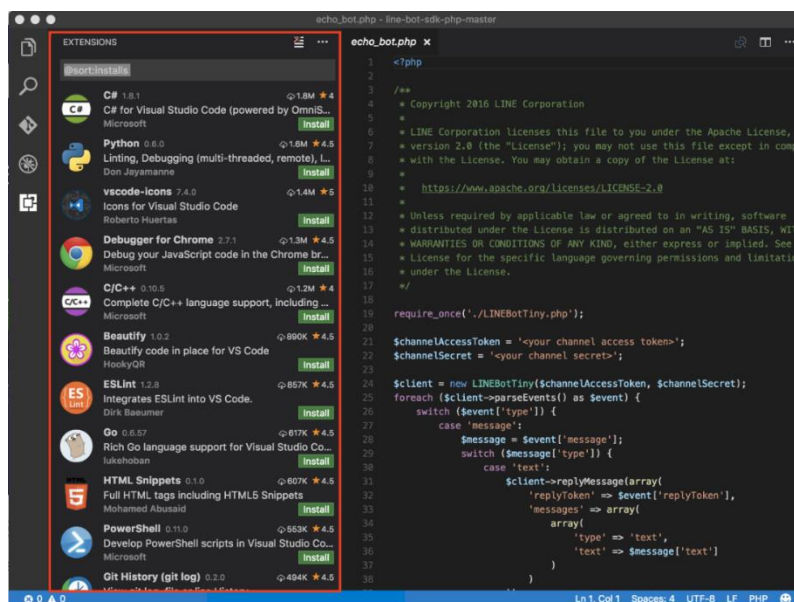
#### 10.1.3. ພາສາທີ່ໃຊ້ຂຽນໂປຣແກຣມ

##### 10.1.3.1. ໂປຣແກຣມ Visual Studio Code

Visual Studio Code ຫຼື ທີ່ຫຼາຍຄົນນິຍົມເອີ້ນຫຍັງເປັນ “vs code” ຂໍບອກກ່ອນວ່າ Editor ໂຕນີ້ມັນອອກມາຕັ້ງແຕ່ 29 ເມສາ ປີ 2015 ແລ້ວ ພັດທະນາຂຶ້ນໂດຍບໍລິສັດຍັກໃຫຍ່ນາມ ໄມໂຄຣຊອບ(Microsoft) ເປັນທັງໂຕແກ້ໄຂ ແລະ ປັບແຕ່ງໂຄດ (code optimized editor) ທີ່ຕັດຄວາມສາມາດມາຈາກ Visual Studio ລຸ້ນປົກກະຕິ (ພວກ GUI designer) ອອກໄປ ເຫຼືອແຕ່ໂຕ editor ຢ່າງດຽວ ສາມາດເຮັດວຽກໄດ້ຂ້າມພອມທັງໝົດ Windows, Mac ແລະ Linux ຊັບພອດພາສາຫຼາຍຮ້ອຍພາສາອີກດ້ວຍ ຊຶ່ງທາງໄມໂຄຊອບເອງນັ້ນໄດ້ເປີດໃຫ້ໃຊ້ຟີອີກດ້ວຍ.

ຄວາມສາມາດຂອງ“vs code” ນັ້ນຈະມີຄວາມສາມາດໃນການເປີດໄດ້ຄືກັບ editor ໂຕອື່ນໆ ເຊັ່ນ sublime, Atom, Notepad++ ນັ້ນເອງ ທັງໝົດເຖິງຄວາມສາມາດໃນການຕິດຕັ້ງເຄື່ອງມືເສີມ (Extension) ໂດຍຮັບຮອງໄດ້ວ່າມີຊັບພອດຢ່າງແນ່ນອນ ເພາະວ່າມັນຖືກພັດທະນາມາໃຫ້ຕອບໂຈດນັກພັດທະນາຫຼາຍທີ່ສຸດ ແມ່ນການດຶຂາຍໜ້າຕາ ໃຫ້ເປັນຮູບແບບທີ່ເຂົ້າໃຈ ແລະ ໃຊ້ງານໄດ້ງ່າຍ ບໍ່ຕ້ອງສຶກສາຫຍັງເພີ່ມເຕີມກໍໃຊ້ງານໄດ້ເລີຍ ສິ່ງທີ່ເຮັດໃຫ້ມັນໂດດເດັ່ນກວ່າໂຕອື່ນໆ ຄືການທີ່ອອກແບບໃຫ້ການຄົ້ນຫາສິ່ງຕ່າງໆ ເຮັດອອກມາໃຫ້ໃຊ້ງານໄດ້ງ່າຍແລະເບິ່ງງ່າຍກວ່າໂຕອື່ນໆທັງໝົດ ເຖິງການທີ່ສ້າງໃຫ້ສາມາດເຊື່ອມຕໍ່ກັບ Git ໄດ້ຢ່າງວ່ອງໄວ ແລະ ງ່າຍດາຍ ມີຟັງຊັນໃນການ commit, push & pull ຢູ່ໃນໂຕ ຫຼື ຈະເບິ່ງ change ຂອງຟາຍທີ່ເກີດຂຶ້ນກໍໄດ້ແບບງ່າຍດາຍບໍ່ຕ້ອງສຶກສາຫຍັງເພີ່ມແຕ່ກໍໃຊ້ງານໄດ້ເລີຍ ([ມາຮູ້ຈັກ “VS Code” ອາວຸດຄູ່ມືສາຍພັດທະນາຂອງໃຜຫຼາຍໆຄົນ](#)

April 3, 2017).



### 10.1.3.2. ພາສາ JavaScript

ພາສາ JavaScript ຫຼື ຕົວຫຍໍ້ JS ເປັນພາສາຂຽນໂປຣແກຣມທີ່ຖືກພັດທະນາ ແລະ ປະຕິບັດຕາມຂໍ້ກຳນົດມາດຕະຖານຂອງ ECMAScript; ພາສາ JavaScript ນັ້ນເປັນພາສາລະດັບສູງ ຄອມພາຍໃນຂະນະທີ່ໂປຣແກຣມຣັນ (JIT) ແລະ ເປັນພາສາຂຽນໂປຣແກຣມແບບຫຼາຍຂະບວນເຊັ່ນ: ການຂຽນໂປຣແກຣມແບບຂັ້ນຕອນ, ການຂຽນໂປຣແກຣມແບບວັດຖຸ, ພາສາ JavaScript ມີໄວຍະກອນທີ່ຄືກັບພາສາ C ໃຊ້ວົງເລັບເພື່ອກຳນົດບ່ອກຂອງຄຳສັ່ງ ນອກຈາກນີ້ JavaScript ຍັງເປັນພາສາທີ່ມີປະເພດຂໍ້ມູນແບບໄດນາມິກ (Dynamic) ເປັນພາສາແບບ Prototype-based ແລະ First-class function.

ພາສາ JavaScript ນັ້ນຖືວ່າເປັນເທັກໂນໂລຊີຫຼັກຂອງການພັດທະນາເວັບໄຊ (World Wide Web) ມັນເຮັດໃຫ້ໜ້າເວັບສາມາດຕອບໂຕ້ກັບຜູ້ໃຊ້ໄດ້ໂດຍທີ່ບໍ່ຈຳເປັນຕ້ອງລືເຟດໜ້າໃໝ່ (Dynamic website) ເວັບໄຊຈຳນວນຫຼາຍໃຊ້ພາສາ JavaScript ສຳລັບຄວບຄຸມການເຮັດວຽກງານທີ່ດ້ານ Client-side ນັ້ນເຮັດໃຫ້ເວັບເບຣາວເຊີຕ່າງໆມີ JavaScript engine ທີ່ໃຊ້ສຳລັບປະມວນຜົນສະຄິບຂອງພາສາ JavaScript ທີ່ຮັບເທິງເວັບບາວເຊີເນື່ອງຈາກພາສາ JavaScript ເປັນພາສາຂຽນໂປຣແກຣມແບບຫຼາຍຮູບແບບເຮັດໃຫ້ມັນຮອງຮັບການຂຽນໂປຣແກຣມທັງແບບ Event-driven, Functional ແລະ ແບບລຳດັບຂັ້ນຕອນມັນມີ Library (APIs) ສຳລັບເຮັດວຽກກັບຂໍ້ຄວາມ, ວັນທີ, Regular expression ແລະ ໂຄງສ້າງຂໍ້ມູນພື້ນຖານຢ່າງ Array ແລະ Map ຫຼື ແມ່ນທັງ Document Object Model (DOM) ຊຶ່ງເປັນ API ທີ່ໂດຍທົ່ວໄປແລ້ວສາມາດໄດ້ເທິງເວັບບາວເຊີ.

ຢ່າງໃດກໍຕາມຕົວຂອງພາສາ JavaScript ເອງບໍ່ໄດ້ມີຟັງຊັນສຳລັບອິນພຸດ/ເອົາພຸດ(I/O) ທີ່ມາກັບພາສາເຊັ່ນ ຟັງຊັນກ່ຽວກັບ Network ວຽກກ່ຽວກັບໄຟລ ຫຼື Library ກ່ຽວກັບກຣາບຟິກໂດຍທົ່ວໄປແລ້ວສິ່ງເຫຼົ່ານີ້ຈະຖືກໃຫ້ມາໂດຍ Host environment (ສະພາບແວດລ້ອມທີ່ໃຊ້ຮັບພາສາ JavaScript) ເຊັ່ນ ເວັບເວັບບາວເຊີ ຫຼື Node.js ຊຶ່ງຈະແຕກຕ່າງກັນອອກໄປ ຕົວຢ່າງເຊັ່ນ: ການຮັບຄ່າໃນເວັບເວັບບາວເຊີຈະຜ່ານຟັງຊັນ prompt ຊຶ່ງເປັນສ່ວນໜຶ່ງຂອງ Browser Object Model (BOM) ຫຼື ຮັບຄ່າຈາກ HTML ຟອມຊຶ່ງເປັນສ່ວນໜຶ່ງຂອງ Document Object Model (DOM) ໃນຂະນະທີ່ເທິງ Node.js ເຮົາສາມາດຮັບຄ່າໄດ້ຈາກ Input/Output Stream ຂອງ Command line ເຖິງແມ່ນວ່າມັນຈະມີຄວາມຄ້າຍຄືກັນລະຫວ່າງພາສາ Java ແລະ JavaScript ເຊັ່ນ: ຊື່ຂອງພາສາໄວຍະກອນ ຫຼື Library ມາດຕະຖານຕ່າງໆ ຢ່າງໃດກໍຕາມທັງສອງພາສາແຕກຕ່າງກັນຢ່າງສິ້ນເຊີງໃນແງ່ຂອງການອອກແບບພາສາ Java ເປັນພາສາທີ່ມີປະເພດຂໍ້ມູນແບບຄົງທີ່ (Static-typing) ໃນຂະນະທີ່ພາສາ JavaScript ມີປະເພດຂໍ້ມູນແບບໄດນາມິກ (Dynamic-typing) ພາສາ Java ຖືກຄອມພາຍເປັນ Byte-code ກ່ອນທີ່ຈະລັນໃນຂະນະທີ່ພາສາ JavaScript ຈະຄອມພາຍໃນຕອນທີ່ໂປຣແກຣມລັນ ພາສາ Java ເປັນພາສາແບບ Class-based ໃນຂະນະທີ່ພາສາ JavaScript ເປັນພາສາແບບ Prototypebased





ຮູບທີ 10 ພາສາ JavaScript

### ກ. ປະຫວັດຄວາມເປັນມາຂອງພາສາ JavaScript

ພາສາ JavaScript ຖືກອອກແບບ ແລະ ສ້າງໂດຍ Brendan Eich ສໍາລັບເປັນພາສາສະຄິບ ທີ່ເຮັດວຽກເທິງເວັບເບຣາວເຊີ Navigator ທີ່ເປັນຜະລິດຕະພັນຂອງບໍລິສັດ Netscape ເພື່ອເຮັດໃຫ້ ໜ້າເວັບທີ່ໃນຕອນທໍາອິດນັ້ນເປັນແບບ Static ສາມາດຕອບໂຕ້ກັບຜູ້ໃຊ້ໄດ້ໂດຍທີ່ບໍ່ຈໍາເປັນຕ້ອງຮິເຟດ ໜ້າໃໝ່ (Dynamic) ເຊັ່ນ: ການສົ່ງຂໍ້ມູນເບື້ອງຫຼັງໄປ Server ແລະ ລໍຖ້າຮັບຜົນຕອບກັບມາດ້ວຍ AJAX; ພາສາ JavaScript ໄດ້ຖືກເປີດຕົວ ແລະ ເປັນສ່ວນໜຶ່ງຂອງເວັບບາວເຊີ Navigator ໃນ ເດືອນກັນຍາ 1995 ໂດຍໃຊ້ຊື່ວ່າ LiveScript ແລະ ໄດ້ປ່ຽນເປັນ JavaScript ໃນອີກສາມເດືອນຕໍ່ມາ.

ໃນເດືອນພະຈິກ 1996 Netscape ໄດ້ສົ່ງພາສາ JavaScript ໄປຍັງ ECMA International ເພື່ອເປັນຈຸດເລີ່ມຕົ້ນສໍາລັບກໍານົດມາດຕະຖານໃຫ້ທຸກເວັບບາວເຊີປະຕິບັດຕາມມາດຕະຖານດັ່ງກ່າວ ເພື່ອໃຫ້ການພັດທະນາ JavaScript engine ເປັນໄປໃນທິດທາງດຽວກັນນັ້ນໃຫ້ເກີດການເປີດຕົວຢ່າງ ເປັນທາງການສໍາລັບຂໍ້ກໍານົດມາດຕະຖານ ECMAScript ໃນເດືອນມິຖຸນາ 1997 ໃນຊ່ວງເວລາຫຼັງ ຈາກນີ້ ບໍລິສັດຕ່າງໆ ທີ່ພັດທະນາເວັບບາວເຊີຕ່າງກໍຍັງພັດທະນາ JavaScript engine ບໍ່ເປັນໄປໃນ ທິດທາງດຽວກັນເທົ່າໃດ ນັ້ນເຮັດໃຫ້ນັກພັດທະນາເວັບຕ້ອງຂຽນໂຄດຫຼາຍເວີຊັນເພື່ອໃຫ້ເຮັດວຽກໄດ້ ໃນທຸກເວັບບາວເຊີຈົນກະທັ້ງໃນເດືອນກໍລະກົດ 2008 ໄດ້ມີການຈັດການປະຊຸມຂຶ້ນທີ່ Oslo ຈາກອົງ ກອນ ແລະ ຝ່າຍຕ່າງໆທີ່ພັດທະນາ JavaScript engine ເຮັດໃຫ້ເກີດຂໍ້ຕົກລົງຂຶ້ນໃນຕົ້ນປີ 2009 ເພື່ອລວບລວມງານທີ່ກ່ຽວຂ້ອງທັງໝົດຂອງພາສາ JavaScript ແລະ ຊຸກຍູ້ພາສາໃຫ້ຢ່າງໄປຂ້າງໜ້າ ນັ້ນເຮັດໃຫ້ເກີດຂໍ້ກໍານົດມາດຕະຖານ ECMAScript ເວີຊັນທີ 5 (ES5) ອອກມາໃນເດືອນທັນວາ 2009 ແລະ ກ່ອນໜ້ານີ້ໃນປີ 2008 Google ໄດ້ເປີດຕົວເວັບບາວເຊີ Chrome ທີ່ມາພ້ອມກັບ V8 JavaScript engine ທີ່ມີແນວຄິດໃນການພັດທະນາແບບຄອມພາຍໃນຕອນທີ່ໂປຣແກຣມລັນ (Just-in-time compilation: JIT) ຊຶ່ງມັນເຮັດວຽກໄດ້ໄວກວ່າຫຼາຍ ເຮັດໃຫ້ຜູ້ພັດທະນາເວັບບາວເຊີອື່ນໆ

ຕ້ອງປັບປຸງ JavaScript engine ຂອງພວກເຂົາໃຫ້ເຮັດວຽກໃນຮູບແບບ JIT ຫຼັງຈາກທີ່ພັດທະນາຕໍ່ເນື່ອງມາອີກຫຼາຍປີ ໃນປີ 2015 ໄດ້ມີການເພີ່ມຄຸນສົມບັດໃໝ່ໆທີ່ຫຼາກຫຼາຍເຂົ້າມາ ຊຶ່ງຖືວ່າເປັນການປ່ຽນແປງເທື່ອສຳຄັນ ແລະ ເຮັດໃຫ້ເກີດຂໍ້ກຳນົດມາດຕະຖານ ECMAScript 2015 ຫຼື ເວີຊັນທີ 6 (ES6) ຈົນກະທັ້ງໃນປີ 2015 ຕອນນີ້ເບິ່ງຄືວ່າພາສາ JavaScript ຈະພັດທະນາມາຈົນເຖິງທີ່ສຸດແລ້ວ ເຮັດໃຫ້ລະຫວ່າງປີ 2016 - 2019 ເວີຊັນໃໝ່ຂອງ ECMAScript ທີ່ຖືກເຜີຍແຜ່ອອກມາໃນແຕ່ລະປີມີການປ່ຽນແປງ ແລະ ເພີ່ມຄຸນສົມບັດພຽງນ້ອຍໆເທົ່ານັ້ນ.

## ຂ. ຄຸນສົມບັດຂອງພາສາ JavaScript

ECMAScript 2015 (ES6) ເປັນພາສາ JavaScript ທີ່ຖືວ່າພັດທະນາມາຈົນເຖິງຈຸດສູງສຸດແລ້ວກໍວ່າໄດ້ ມັນຖືກເຜີຍແຜ່ໃນເດືອນມິຖຸນາ 2015 ຊຶ່ງໃນເວີຊັນນີ້ ໄດ້ເພີ່ມໄວຍະກອນໃໝ່ຂອງພາສາຫຼວງຫຼາຍເຊັ່ນ: ການສ້າງຄາດດ້ວຍຄຳສັ່ງ class ການສ້າງໂມດູນ ແລະ ໃຊ້ງານມັນດ້ວຍຄຳສັ່ງ import ແລະ export ແລະ ຄຳສັ່ງສຳລັບປະກາດຕົວປ່ຽນ let ແລະ ປະກາດຄ່າຄົງທີ່ const ຊຶ່ງເຮັດໃຫ້ຕົວປ່ຽນສາມາດມີຂອບເຂດໃນບລັອກທີ່ມັນຖືກສ້າງຂຶ້ນໄດ້ ແລະ ສິ່ງອື່ນໆ ທີ່ຖືກເພີ່ມເຂົ້າມາເປັນຈຳນວນຫຼາຍເຊັ່ນ: Map, Set, WeakMap, Promise, Reflection, Proxies, Template string ແລະ ອື່ນໆ.

ໃນເດືອນມິຖຸນາ 2016 ໄດ້ມີການເປີດຕົວເວີຊັນ 7 ຫຼື ECMAScript 2016 (ES7) ໄດ້ມີການເພີ່ມຕົວດຳເນີນການຍົກກຳລັງ \*\* (ທີ່ກ່ອນໜ້ານີ້ເຮົາຈະໃຊ້ຜ່ານພັງຊັນ Math.pow) ຄຳສັ່ງ await async ສຳລັບການຂຽນໂປຣແກຣມທີ່ເຮັດວຽກບໍ່ພ້ອມກັນ ແລະ ພັງຊັນ includes ຂອງອາເລ ແລະ ໃນປະຈຸບັນພາສາ JavaScript ຖືກພັດທະນາມາຈົນເຖິງ ECMAScript 2020 (ES11) ຊຶ່ງມີການປ່ຽນແປງທີ່ເພີ່ມຂຶ້ນບໍ່ເທົ່າໃດຫຼັງຈາກ ES7

## ຄ. JavaScript engine ແມ່ນຫຍັງ?

JavaScript engine ຄືໂປຣແກຣມຄອມພິວເຕີທີ່ໃຊ້ສຳລັບປະມວນຜົນໂຄດຂອງພາສາ JavaScript ຊຶ່ງ JavaScript engine ໃນຊ່ວງເລີ່ມຕົ້ນເປັນພຽງແຕ່ຕົວປ່ຽນພາສາ (Interpreter) ເທົ່ານັ້ນ ແຕ່ໃນປະຈຸບັນໄດ້ມີການພັດທະນາມາໃຫ້ຢູ່ໃນຮູບແບບຂອງຄອມພາຍເລີທີ່ມີການຄອມພາຍໃນຕອນທີ່ໂປຣແກຣມລັນ (Just-in-time compilation: JIT) ເພື່ອເພີ່ມປະສິດທິພາບການເຮັດວຽກງານຂອງໂປຣແກຣມ ໂດຍທົ່ວໄປແລ້ວ JavaScript engine ຈະຖືກພັດທະນາໂດຍຜູ້ພັດທະນາເວັບບາວເຊີທີ່ປະຕິບັດຕາມຂໍ້ກຳນົດມາດຕະຖານຂອງ ECMAScript ([ແນະນຳພາສາ JavaScript](http://marcuscode.com/lang/javascript/introducing-to-javascript) <http://marcuscode.com/lang/javascript/introducing-to-javascript> 28 July 2020).

### 10.1.3.3. Reactjs

React ເປັນເທັກໂນໂລຢີໜຶ່ງທີ່ມາແຮງຫຼາຍໂດຍສ້າງມາຈາກພື້ນຖານແນວຄວາມຄິດແບບ MVC (Model View Controller) ຊຶ່ງໝາຍຄວາມວ່າ React ມີໜ້າທີ່ຈັດການກັບ Model ຫຼື View ແຕ່ສ່ວນໃຫຍ່ຈະເປັນ View ກ່ອນໜ້ານັ້ນເວລາຈະຂຽນໜ້າເວັບເຮົາກໍຈະເຮັດຜ່ານ HTML ມີການໃຊ້ CSS ໃນການປັບປ່ຽນໜ້າຕາຂອງ UI ແຕ່ໃນ React ຖ້າຈະສ້າງໜ້າເວັບຂຶ້ນມາໄດ້ນັ້ນ ເຮົາຈະໃຊ້ເປັນ Component ປຸງບໄດ້ວ່າ Component ເປັນ Block ສ່ວນຍ່ອຍຂອງເວັບເຮົາທີ່ຈະສ້າງອອກ

ມາ. (ມາທຳຄວາມຮູ້ຈັກ React <https://www.mindphp.com/ບທຣີຍນອນໂລນ/tutorial-reactjs/4883-react.html> 13 May 2017).

### ສະຫລຸບງ່າຍໆດັ່ງນີ້

- React ເປັນ Javascript Libraly ສ້າງ ແລະ ພັດທະນາຂຶ້ນໂດຍ Facebook ຂຽນໄດ້ແຕ່ UI ເທົ່ານັ້ນ ແລະ ເປີດໃຫ້ໃຊ້ຟຣີ
- React ໃຊ້ໂຄດ HTML , CSS ແລະ Javascript
- React ມີ 3 ຄອນເຊບທີ່ເຮົາຕ້ອງຮຽນຮູ້ຄື Component State ແລະ Props

#### ➤ ຂໍ້ດີ ແລະ ຂໍ້ເສຍຂອງ React

##### ກ. ຂໍ້ດີ

- Component ເຂົ້າໃຈງ່າຍເຮົາສາມາດຮຽນຮູ້ໄດ້ດ້ວຍຕົວເອງ
- Tool ຫຼາຍພຽງ React ຢ່າງດຽວກໍ່ສາມາດຂຽນເວັບໄດ້ທັງເວັບແລ້ວໂດຍບໍ່ຕ້ອງຫາ Tool ເພີ່ມເຕີມ ແລະ ຍັງມີ Tool ພັດທະນາອອກມາຢູ່ເລື້ອຍໆສາມາດເຮັດ App ໄດ້ React ມີເຄື່ອງມືໜຶ່ງທີ່ຊື່ວ່າ React Native ເປັນການຂຽນ JavaScript ແລ້ວແປງເປັນ App ແບບ Native ໄດ້ທັງເທິງ Android ແລະ iOS

##### ຂ. ຂໍ້ເສຍ

- ຕ້ອງມີພື້ນຖານໃນ Javascript ໃນລະດັບໜຶ່ງ ຄົນທີ່ສົນໃຈຮຽນກໍຈະລຳບາກໜ້ອຍໜຶ່ງຖ້າບໍ່ໄດ້ມີພື້ນຖານ Javascript ອາດຈະຕ້ອງໃຊ້ເວລາໜ້ອຍໜຶ່ງ
- Documentation ອ່ານຍາກ React ມີ Documentation ທີ່ຍັງບໍ່ຄ່ອຍດີໃນອະນາຄົດອາດຈະມີການພັດທະນາອີກ

#### 10.1.3.4. Nodejs

Node.js ເປັນ open-source ແລະ cross-platform JavaScript runtime environment ທີ່ກຳລັງໄດ້ຮັບຄວາມນິຍົມສູງ ໂດຍທົ່ວໄປເຮົາຈະໃຊ້ JavaScript ໃນຝັ່ງ client ແຕ່ Node.js ເຮັດໃຫ້ເຮົາໃຊ້ JavaScript ໃນຝັ່ງ Server ໄດ້ດ້ວຍຊື່ງ Node.js ສາມາດ run ໄດ້ເທິງ platform ທີ່ຫຼາກຫຼາຍທັງ Windows, Linux, Unix, Mac OS X ແລະ ອື່ນໆ.



## ຮູບທີ 12 Nodejs

ຈຸດເດັ່ນທີ່ສຸດຂອງ Node.js ຄືມັນເຮັດວຽກແບບ asynchronous ຜູ້ອ່ານອາດຈະສົງໃສວ່າ ແລ້ວມັນດີແນວໃດ? ລອງມາເບິ່ງຕົວຢ່າງການຈັດການໄຟລເມື່ອມີການຮ້ອງຂໍຈາກ client ມາທີ່ server ຂອງ PHP ກັບ Node.js ທຽບກັນເບິ່ງເພື່ອໃຫ້ເຂົ້າໃຈຫລາຍຂຶ້ນ

➤ ກໍລະນີ PHP ເມື່ອມີການຮ້ອງຂໍເຂົ້າມາມັນຈະເຮັດດັ່ງນີ້:

1. ສົ່ງ task ໄປທີ່ລະບົບໄຟຂອງຄອມພິວເຕີ
2. ລໍຖ້າຈົນກະທັ້ງລະບົບໄຟລເປີດ ແລະ ອ່ານໄຟລສໍາເລັດ
3. ສົ່ງເນື້ອຫາຂອງໄຟລກັບມາໃຫ້ client
4. ພ້ອມສໍາລັບຮັບຄໍາຮ້ອງຂໍຖັດໄປ

➤ ກໍລະນີ Node.js ເມື່ອມີການຮ້ອງຂໍເຂົ້າມາມັນຈະເຮັດດັ່ງນີ້:

1. ສົ່ງ task ໄປທີ່ລະບົບໄຟຂອງຄອມພິວເຕີ
2. ພ້ອມສໍາລັບຮັບຄໍາຮ້ອງຂໍຖັດໄປ
3. ພໍລະບົບໄຟລເປີດ ແລະ ອ່ານໄຟລສໍາເລັດ server ຈະສົ່ງເນື້ອຫາຂອງໄຟລກັບມາໃຫ້ client

ຈາກຕົວຢ່າງດ້ານເທິງຈະເຫັນວ່າ Node.js ຈະຕັດຂັ້ນຕອນການລໍຖ້າຖິ້ມແລ້ວໄປເຮັດຄໍາຮ້ອງຖັດໄປເລີຍ ທີ່ເປັນແບບນີ້ເພາະ Node.js ຈະ run ແບບ single-threaded ແລະ ໃນ Library ມາດຕະຖານກໍຈະມີເຊັດຂອງ asynchronous I/O primitives ທີ່ຊ່ວຍປ້ອງກັນໂຄດ JavaScript ຈາກການ blocking ເຮັດໃຫ້ລະບົບຄ່ອງແຄ້ວ ແລະ ມີປະສິດທິພາບຫຼາຍຂຶ້ນ

Synchronous vs Asynchronous	
<p>Synchronous ຄືການ run ໂຄດຕາມລຳດັບທີ່ເຮົາຂຽນໄວ້ ເຊັ່ນ</p> <pre>alert(1); alert(2); alert(3);</pre> <p>ຜົນລັບທີ່ໄດ້ຄືໜ້າຈໍຈະສະແດງຜົນ 1 ຈາກນັ້ນຈຶ່ງສະແດງຜົນ 2 ແລ້ວສຸດທ້າຍຈຶ່ງສະແດງຜົນ 3 ຕາມລຳດັບ</p>	<p>Asynchronous ຄືການຮັບໂຄດທີ່ບໍ່ຈຳເປັນຕ້ອງເປັນໄປຕາມລຳດັບທີ່ເຮົາຂຽນໄວ້ ເຊັ່ນ:</p> <pre>alert(1); setTimeout(() =&gt; alert(2), 0); alert(3);</pre> <p>ໃນຕົວຢ່າງນີ້ໂຄດ alert(2) ໃຊ້ເວລາດຳເນີນການດົນກວ່າ ເຮັດໃຫ້ຜົນລັບທີ່ໄດ້ຄືໜ້າຈໍຈະສະແດງຜົນ 1 ຈາກນັ້ນຈຶ່ງສະແດງຜົນ 3 ແລ້ວສຸດທ້າຍຈຶ່ງສະແດງຜົນ 2</p>
Blocking vs Non-blocking	
<p>Blocking ໝາຍເຖິງການທີ່ເຮົາບໍ່ສາມາດດຳເນີນການຕໍ່ໄປໄດ້ຈົນກວ່າຕົວດຳເນີນການທີ່ກຳລັງລົ້ນຢູ່ຈະສໍາເລັດເສຍກ່ອນ ເຊັ່ນ</p>	<p>Non-blocking ໝາຍເຖິງການໂຕທີ່ດຳເນີນການສາມາດເຮັດຄໍາສັ່ງຖັດໄປໄດ້ເລີຍໂດຍບໍ່ຕ້ອງລໍຖ້າໃຫ້ຄໍາສັ່ງເດີມເຮັດສໍາເລັດກ່ອນເຊັ່ນ:</p>

<pre> alert(1); var value = localStorage.getItem('foo'); alert(2); </pre> <p>ຄຳສັ່ງ localStorage ຈະເປັນຕົວ blocking ເຮັດໃຫ້ບໍ່ສາມາດເຮັດຄຳສັ່ງ alert(2) ໄດ້ຈົນກວ່າມັນຈະດຳເນີນການສຳເລັດ ດັ່ງນັ້ນຜົນລັບທີ່ໄດ້ຄືໜ້າຈໍຈະສະແດງຜົນ 1 ຈາກນັ້ນຈຶ່ງລໍຖ້າຈົນກວ່າຄຳສັ່ງ localStorage ຈະສຳເລັດຈຶ່ງສະແດງຜົນ 2</p>	<pre> alert(1); fetch('example.com').then(() =&gt; alert(2)); alert(3); </pre> <p>ໃນຕົວຢ່າງນີ້ ຄຳສັ່ງ fetch ເປັນ non-blocking operation ດັ່ງນັ້ນຜົນລັບທີ່ໄດ້ຄືໜ້າຈໍຈະສະແດງຜົນ 1 ຈາກນັ້ນຈຶ່ງສະແດງຜົນ 3 ແລ້ວພໍຄຳສັ່ງ fetch ສຳເລັດກໍສະແດງຜົນ 2</p>
---	---

## ➤ ປະຫວັດ Node.js

ແຕ່ເດີມພາສາ JavaScript ຖືກພັດທະນາມາເພື່ອໃຊ້ສຳລັບ Browser ທີ່ຊື່ Netscape Navigator ໃນປີ 1995 ໃນຕອນນັ້ນ Netscape ຕັ້ງໃຈຈະຂາຍ Web Server ທີ່ມີ environment ຊື່ Netscape LiveWire ຊຶ່ງສາມາດສ້າງ dynamic page ໂດຍໃຊ້ JavaScript ທາງຝັ່ງ server ອີກດ້ວຍ ແຕ່ໜ້າເສຍດາຍທີ່ Netscape LiveWire ບໍ່ປະສົບຄວາມສຳເລັດ ແລະ ການໃຊ້ JavaScript ທາງຝັ່ງ Server ກໍບໍ່ໄດ້ຮັບຄວາມນິຍົມເລີຍຈົນກະທັ້ງ Node.js ຖືກກຳເນີດຂຶ້ນມາ.

ສິ່ງທີ່ເຮັດໃຫ້ Node.js ເປັນທີ່ນິຍົມຂຶ້ນມາຄືການທີ່ມັນມາໃນຊ່ວງເວລາທີ່ເໝາະສົມ ເມື່ອທຽບກັບ JavaScript ທີ່ເກີດມາຕັ້ງແຕ່ປີ 1995 ແລ້ວ Node.js ຫາກໍເກີດມາເມື່ອປີ 2009 ເທົ່ານັ້ນ ຕ້ອງຂອບໃຈ "Web 2.0" applications ເຊັ່ນ Flickr, Gmail ແລະ ອື່ນໆທີ່ສະແດງໃຫ້ໂລກຮູ້ວ່າເວັບສະໄໝໃໝ່ຄວນຫ້າຕາເປັນຢ່າງໃດ

ບໍ່ດົນກ່ອນທີ່ Node.js ຈະເກີດນັກພັດທະນາເບື້ອງໜຶ່ງ browser ຊຶ່ງດັ່ງທັງຫຼາຍແຂ່ງຂັນກັນເຮັດວຽກຢ່າງໜັກເພື່ອຈະໃຊ້ JavaScript ໃຫ້ໄດ້ດີທີ່ສຸດ ແລະ ຫາທາງເຮັດໃຫ້ JavaScript ສາມາດລັນໄດ້ໄວຫລາຍຂຶ້ນເພື່ອໃຫ້ຜູ້ໃຊ້ງານໄດ້ຮັບປະສິດທິພາບທີ່ດີທີ່ສຸດ ຊຶ່ງຜົນຈາກການແຂ່ງຂັນນີ້ເຮັດໃຫ້ເກີດການພັດທະນາ Chrome V8 (open-source JavaScript engine ຂອງ The Chromium Project) ຂຶ້ນມາ ແລະ Node.js ກໍໃຊ້ engine ນີ້ເອງ

- ແຕ່ການທີ່ Node.js ເປັນທີ່ນິຍົມຂຶ້ນມາບໍ່ແມ່ນແຕ່ວ່າມັນມາຖືກທີ່ຖືກເວລາເທົ່ານັ້ນ ແຕ່ມັນໄດ້ສະແດງໃຫ້ເຫັນແລ້ວວ່າ ການອອກແບບ ແລະ ແນວຄິດຂອງມັນຊ່ວຍນັກພັດທະນາທັງຫຼາຍໃຫ້ສາມາດໃຊ້ JavaScript ທາງຝັ່ງ server ໄດ້ງ່າຍຂຶ້ນຫລາຍອີກດ້ວຍ. (Nodejs ແມ່ນຫຍັງ [https://expert-programming-tutor.com/tutorial/article/A06\\_NodeJS01\\_HelloWorld.php](https://expert-programming-tutor.com/tutorial/article/A06_NodeJS01_HelloWorld.php) 13 May 2017).

### 10.1.3.5. ExpressJS

Express.js ເປັນ Web Application Framework ຊຶ່ງດັ່ງທີ່ໄດ້ຮັບຄວາມນິຍົມຫຼາຍສຳລັບເຮັດວຽກເທິງ platform ຂອງ Node.js ຊຶ່ງເປັນ Server ຕົວໜຶ່ງໂດຍທັງ Express.js ແລະ Node.js ຕ່າງກໍໃຊ້ພາສາ JavaScript ໃນການພັດທະນາ ຖ້າເປັນ Web Application Framework ໃນສະໄໝກ່ອນ ຄົນທີ່ພັດທະນາຈະຕ້ອງມີຄວາມຮູ້ຫຼາຍກວ່າ 1 ພາສາ, ພາສາທີ່ເຮັດວຽກທາງຝັ່ງ Server ຢ່າງ

PHP ຫຼື ASP ແລະ ພາສາທີ່ເຮັດວຽກທາງຝັ່ງ Client ຢ່າງ JavaScript ເພື່ອຫຼຸດຄວາມຫຍຸ້ງຍາກທັງ ໝົດເຖິງເວລາໃນການຕ້ອງຮຽນຮູ້ຫຼາຍໆພາສາເຮັດໃຫ້ເກີດ Node.js ກັບ Express.js ພຽງແຕ່ມີຄວາມ ຮູ້ JavaScript ກໍສາມາດຂຽນໄດ້ທັງ Server ແລະ Client ນອກຈາກນີ້ ຖ້າໃຜເຄີຍຂຽນ JavaScript ຈະຮູ້ວ່າມັນມີການຕອບສະໜອງທີ່ວ່ອງໄວແນ່ນອນວ່າ Express.js ກໍຍົກເອົາມາເປັນຂໍ້ເດັ່ນໃນເລື່ອງ ຄວາມໄວ ໃນເລື່ອງການຮຽນຮູ້ການຂຽນ Express.js ຈະໃຊ້ຮູບແບບທີ່ງ່າຍໃນການຮຽນຮູ້ຫລາຍທີ່ສຸດ ສໍາລັບການພັດທະນາ Express.js ໃນເວັບໄຊຈະເວົ້າເຖິງການໃຊ້ Routing (ການກຳນົດເສັ້ນທາງຂອງ ລະບົບ) ແລະ Middleware (ການຮັບສົ່ງຂໍ້ມູນຂອງລະບົບ) ສາມາດຂຽນໄດ້ໃນຮູບແບບ MVC ສ່ວນ ການເຊື່ອມຕໍ່ກັບຖານຂໍ້ມູນສາມາດໃຊ້ MongoDB ຫຼື ຈະໃຊ້ MySQL ກໍໄດ້ສໍາລັບນາມສະກຸນຂອງ ໄຟລຕີ .js ຂະນະນີ້ໄດ້ພັດທະນາມາເຖິງເວີຊັນທີ່ 4.x ແລ້ວ.

#### 10.1.3.6. Version Control (Git ແລະ GitHub)

ເນື່ອງຈາກທຸກມື້ນີ້ແມ່ນໃຊ້ Git ໃນການເຮັດວຽກງານຢູ່ແລ້ວ ແລະ ບາງເທື່ອກໍຕ້ອງສອນເລື່ອງ ນີ້ໃຫ້ກັບສະມາຊິກໃໝ່ພາຍໃນທີມ ກໍເລີຍຕັດສິນໃຈຂຽນບົດຄວາມນີ້ຂຶ້ນມາ ເພື່ອທີ່ຈະໄດ້ແຊຄວາມຮູ້ໃຫ້ ກັບ Developer ທີ່ສົນໃຈໃນ Git ແຕ່ຍັງບໍ່ເຂົ້າໃຈ ຫຼື ບໍ່ຮູ້ວ່າຈະເຮັດຄວາມເຂົ້າໃຈແບບໃດດີ ເພາະທຸກ ມື້ນີ້ໂລກຂອງ Developer ນັ້ນໄປໄວເຫຼືອເກີນ ຫຼາຍໆຢ່າງຖ້າເຮົາບໍ່ຮູ້ຈັກ ຫຼື ໃຊ້ບໍ່ເປັນກໍອາດຈະເຮັດ ໃຫ້ພາດໂອກາດດີໆໃນເສັ້ນທາງນີ້ກໍເປັນໄດ້

##### 1) ຄວາມສໍາຄັນຂອງ Version Control

ລອງຄິດເບິ່ງວ່າຖ້າຕ້ອງພັດທະນາໂປຣເຈກຂະໜາດໃຫຍ່ທີ່ມີ Developer 4 ຄົນທີ່ກໍາລັງຂຽນ ໂຄ້ດໃນໂປຣເຈກນີ້ຢູ່ ຈະໃຊ້ວິທີໃດເພື່ອເອົາໂຄ້ດທີ່ແຕ່ລະຄົນຂຽນມາທັງຫມົດເຂົ້ານຳກັນໃນໂປຣເຈກ

ວິທີເກົ່າທີ່ໃຊ້ກັນກໍຄື ກ່ອນໂປຣເຈກຈາກແຕ່ລະຄົນມາທັງຫມົດໄວ້ໃນເຄື່ອງດຽວກັນ ແລ້ວນັ່ງ ລວມກັນ ມີຄົນໜຶ່ງເຮັດໜ້າທີ່ເປີດໂຄ້ດຂອງແຕ່ລະຄົນຂຶ້ນມາ ສົມໝຸດວ່າຄົນໆນັ້ນຄື ທ້າວ A ແລະ ທ້າວ B ເປັນຄົນຂຽນໂຄ້ດທີ່ກໍາລັງຈະທັງຫມົດໄວ້ໃນໂປຣເຈກດຽວກັນ ທ້າວ A ກໍເລີຍຕ້ອງຖາມ ທ້າວ B ວ່າຂຽນໂຄ້ດບ່ອນໃດ ແລ້ວຄ່ອຍເອົາໄປແປຂທັງຫມົດໄວ້ໃນໂປຣເຈກຫຼັກ

ບັນຫາເກົ່າໆທີ່ເກີດຂຶ້ນຢູ່ປະຈຳກໍຄື ໂຄ້ດທີ່ ທ້າວ B ໄປແກ້ໄຂພັດໄປທັບຊ້ອນກັບ ທ້າວ C ເພາະທ້າວ C ກໍແກ້ໄຂໂຄ້ດຈຸດນັ້ນຄືກັນ ກາຍເປັນວ່າ ທ້າວ B ກໍຕ້ອງໄປເອີ້ນ ທ້າວ C ມານັ່ງຖາມ ເພື່ອບອກໃຫ້ ທ້າວ A ແກ້ໄຂໃຫ້ໂຄ້ດຂອງ ທ້າວ B ແລະ ທ້າວ C ເຮັດວຽກຮ່ວມກັນໄດ້

ຍັງບໍ່ລວມໄປເຖິງກໍລະນີທີ່ ທ້າວ B ຈຶ່ງບໍ່ໄດ້ວ່າຕົວເອງແກ້ໄຂໂຄ້ດຈຸດໃດແດ່ ເພາະພິເຈດີທີ່ເຮັດ ນັ້ນໃຊ້ເວລາຫຼາຍມື້ ແລະ ຂຽນໂຄ້ດຫຼາຍບັນທັດ ກາຍເປັນວ່າຕ້ອງນັ່ງລວມກັນເພື່ອໄລ່ເບິ່ງໂຄ້ດວ່າ ພາດຈຸດໃດ ຫຼື ບໍ່ ຊຶ່ງລວມໄປເຖິງບັນຫາອື່ນໆທີ່ຈະຕາມມາອີກຫລວງຫລາຍ

ສະນັ້ນ ຈຶ່ງເຮັດໃຫ້ເກີດຂຶ້ນທີ່ເອີ້ນວ່າ Version Control ຂຶ້ນມາເພື່ອຄວບຄຸມການປ່ຽນແປງ ຂອງໂຄ້ດໃນໂປຣເຈກ ໂດຍປະໂຫຍດທີ່ເຫັນໄດ້ແຈ້ງຂອງ Version Control ຈະມີດັ່ງນີ້:

- ເກັບປະຫວັດການແກ້ໄຂໂຄດໄວ້ທຸກເທື່ອ ແລະ ຮູ້ໄດ້ວ່າໂຄ້ດຈຸດໃດໃຜເປັນຄົນເພີ່ມເຂົ້າມາ ຫຼື ແກ້ໄຂ

- ຊ່ວຍລວມໂຄ້ດຈາກຫຼາຍໆຄົນເຂົ້ານຳກັນໃຫ້ງ່າຍຂຶ້ນ ເບິ່ງໄດ້ວ່າໂຄ້ດເດີມຄືຫຍັງ ແລະ ແກ້ໄຂ ເປັນຫຍັງ
- ເມື່ອເກີດບັນຫາກໍສາມາດຕິດຕາມເບິ່ງປະຫວັດການແກ້ໄຂໂຄ້ດໃນແຕ່ລະໄຟລ ແຕ່ລະບັນທັດໄດ້ ງ່າຍ
- ຊ່ວຍໃຫ້ສາມາດຈັດການໂປຣເຈກໄດ້ຢ່າງເປັນ ເບິ່ງໂຄ້ດແຕ່ລະສ່ວນເປັນຟີເຈີ ບໍ່ຊຽນໂຄ້ດຂ້າມຟີ ເຈີໄປມາໃນໂຄ້ດຊຸດດຽວກັນ
- ເປັນ Backup ໄປໃນຕົວ ບໍ່ຕ້ອງຍ້ານເວລາໂຄ້ດມີບັນຫາແລ້ວຕ້ອງ Rollback ກັບໄປໃຊ້ໂຄ້ດ ຊຸດເກົ່າ ແລະ ໃຊ້ພື້ນທີ່ໃນການເກັບຂໍ້ມູນນ້ອຍເມື່ອທຽບກັບການ Backup ແບບເກັບທັງໂປຣເຈກ ໄວ້ທຸກເທື່ອບ່ອນເຮັດວຽກ Backup
- ສາມາດ Track ການເຮັດວຽກງານຂອງທຸກຄົນພາຍໃນທີມໄດ້ຈາກ History

## 2) ການເຮັດວຽກຂອງ Git

ທີ່ຈິງແລ້ວມັນກໍຄືການເຮັດວຽກງານແບບ Distributed Version Control ລອງຄິດເຖິງພາບວ່າ ມີ Server ກາງເຄື່ອງໜຶ່ງທີ່ລໍຖ້າເກັບຂໍ້ມູນຈາກຜູ້ໃຊ້ແຕ່ລະເຄື່ອງກ່ອນ

Server ກາງເອີ້ນວ່າ Remote 1 ສ່ວນ Developer ທີ່ເຮັດໂປຣເຈກນີ້ຢູ່ຈະເອີ້ນວ່າ Computer 1 ແລະ 2 ຊຶ່ງແຕ່ລະຄົນກໍເຮັດຄົນລະ Feature ຢູ່ຊຶ່ງ Feature ຂອງແຕ່ລະຄົນນັ້ນກໍຈະຖືກເກັບໄວ້ທີ່ Remote 1, ອາດຈະເບິ່ງຄືວ່າມັນເປັນລະບົບ Server ກາງທີ່ຄອຍຖ້າ Backup ຂໍ້ມູນຂອງທຸກຄົນຢູ່ ຕະຫລອດເວລາແຕ່ໃນຄວາມເປັນຈິງ Git ນັ້ນມີຫຍັງຫຼາຍກວ່ານີ້ອີກຫຼາຍຢ່າງ, Git ອອກແບບມາໃຫ້ ເຮັດວຽກກະຈາຍແບບບໍ່ມີສູນກາງ ທຸກເຄື່ອງເຮັດວຽກເປັນ VCS ດ້ວຍຕົວເອງໄດ້. ນັ້ນໝາຍຄວາມວ່າ ບໍ່ຈຳເປັນຕ້ອງມີ Server ກາງກໍໄດ້ ສາມາດໃຊ້ເຄື່ອງສ່ວນຕົວເຮັດເປັນ VCS ໄດ້ເລີຍ ແຕ່ຖ້າຕ້ອງເຮັດ ວຽກຮ່ວມກັນຫຼາຍໆເຄື່ອງ ກໍຕ້ອງໃຊ້ Server ເປັນຕົວກາງໃນການຂໍ້ມູນ, ຊຶ່ງຈະເຮັດໃຫ້ຂໍ້ມູນຂອງເຮົາ ບໍ່ຜູກຂາດກັບ Server ຈົນເກີນໄປ ໃນເວລາທີ່ Server ກາງມີບັນຫາ ຫຼື ວ່າເຮັດວຽກແບບ Offline ເຮົາກໍຍັງຄົງເຮັດວຽກໄດ້ຢູ່ ໂດຍໃຊ້ຂໍ້ມູນຈາກ VCS ພາຍໃນເຄື່ອງຕົວເອງ ພໍເຊື່ອມຕໍ່ກັບ Server ກາງ ກໍຄ່ອຍ Sync ຂໍ້ມູນພາຍຫຼັງໄດ້ ແລະ ເມື່ອທຸກໆເຄື່ອງເຮັດວຽກເປັນ VCS ຢູ່ແລ້ວຈຶ່ງເຮັດໃຫ້ເຮົາ ສາມາດມີ Server ກາງຫຼາຍກວ່າ 1 ເຄື່ອງໄດ້ເຊັ່ນກັນ

## 3) Server ສຳລັບໃຫ້ບໍລິການ Git (Version Control Repository Hosting Service)

ການໃຊ້ Git ໃຫ້ເກີດປະໂຫຍດທີ່ສຸດຕ້ອງມີ Server ທີ່ເຮັດໜ້າທີ່ເປັນ VCS ຊຶ່ງໃນປະຈຸ ບັນນີ້ກໍມີທາງເລືອກຫລວງຫລາຍບໍ່ວ່າຈະໃຊ້ບໍລິການຈາກເວັບຕ່າງໆ ຫຼື ຕັ້ງ Server ຂຶ້ນມາເອງ

ໃນກໍລະນີເລີ່ມຕົ້ນ ຂໍແນະນຳໃຫ້ໃຊ້ Server ທີ່ໃຫ້ບໍລິການ Git ໄປເລີຍ ເພາະເຮົາບໍ່ຕ້ອງ ຈັດການຫຍັງຫລາຍ ພຽງແຕ່ສະໝັກກຳເຂົ້າໄປໃຊ້ງານໄດ້ເລີຍ ໂດຍເວັບໄຊທີ່ນິຍົມກໍຈະມີດັ່ງນີ້:

GitHub, Bitbucket, Gitlab. ຊຶ່ງເວັບເຫຼົ່ານີ້ຈະມີໃຫ້ບໍລິການທັງແບບ Public (ໃຜໆກໍເຂົ້າມາເບິ່ງໂຄດຂອງເຮົາໄດ້) ແລະ Private (ສະເພາະຄົນທີ່ເຮົາອະນຸຍາດເທົ່ານັ້ນ) ຖ້າຢາກເຮັດເປັນ Open Source ຢູ່ແລ້ວກໍແນະນຳ GitHub

#### 4) ຄຳຕ່າງໆທີ່ຈະຕ້ອງຮູ້ຈັກເມື່ອໃຊ້ງານ Git

##### ກ. Repository

ເວລາທີ່ເຮົາພັດທະນາໂປຣແກຣມເຮົາຈະຕ້ອງສ້າງສິ່ງທີ່ເອີ້ນວ່າ Project ຊຶ່ງການສ້າງ Project ສຳລັບໃຊ້ງານ Git ຈະເອີ້ນວ່າ Repository

ຊຶ່ງ Repository ຂອງ Git ແທ້ໆແລ້ວກໍຄື Folder ທີ່ໃຊ້ເກັບຂໍ້ມູນນັ້ນເອງ ຢາກຈະເກັບຫຍັງໄວ້ໃນນັ້ນກໍຍັດເຂົ້າໄປໄດ້ເລີຍ (ໃນຄວາມເປັນຈິງ 1 Repository ສາມາດເກັບ Project ເທົ່າໃດກໍໄດ້ຕາມທີ່ຕ້ອງການແຕ່ສ່ວນໃຫຍ່ກໍມີເກັບ Project 1 ຕົວຕໍ່ 1 Repository)

ເວລາທີ່ຜູ້ອ່ານເປີດເບິ່ງໂຄດຕ່າງໆຂອງນັກພັດທະນາເທິງ GitHub (ຍົກຕົວຢ່າງເຊັ່ນ <https://github.com/google/volley>) ໃນແຕ່ລະໂປຣເຈກທີ່ເຮົາເຫັນທີ່ເອີ້ນວ່າ Repository

##### ຂ. Clone

ເວລາທີ່ຜູ້ອ່ານມີ Repository ຢູ່ເທິງ Remote ຢູ່ແລ້ວ ແລະ ຕ້ອງການ Sync ມາລົງເຄື່ອງຂອງເຮົາ ເຮົາຈະຕ້ອງເຮັດສິ່ງທີ່ເອີ້ນວ່າ Clone Repository ຫຼື ກໍຄືການກ່ອບ Repository ຈາກ Remote ມາລົງເຄື່ອງນັ້ນເອງ

##### ຄ. Commit

ເວລາທີ່ມີຂໍ້ມູນທີ່ແກ້ໄຂສຳເລັດແລ້ວ (ໂຄດທີ່ຊຽນຄຳສັ່ງບາງຢ່າງສຳເລັດແລ້ວ) ແລ້ວຢາກຈະກໍານົດ Backup ເກັບໄວ້ໃນ VCS ຈະຮຽກກັນວ່າ Commit ຊຶ່ງການ Commit ຈະສາມາດເລືອກໄດ້ວ່າຈະເອົາໄຟລໄດອີກ (ບໍ່ຈຳເປັນຕ້ອງເລືອກທຸກໄຟລ)

ຊຶ່ງເບື້ອງທຶນຂອງ Commit ກໍຄືໃນແຕ່ລະເທື່ອທີ່ເຮັດວຽກ Commit ມັນຈະຈື່ແຕ່ວ່າມີຈຸດໃດຂອງຂໍ້ມູນທີ່ຖືກປ່ຽນແປງໄປເລື້ອຍໆ ດັ່ງນັ້ນ ໃນແຕ່ລະ Commit ມັນຈະບໍ່ມີໄຟລຂໍ້ມູນສະບັບເຕັມ ແຕ່ມັນສາມາດຍ້ອນເບິ່ງ History ໄດ້ວ່າມີການແກ້ໄຂຫຍັງແດ່ ເຮັດໃຫ້ຮູ້ວ່າໃນ Commit ນັ້ນໆແຕ່ລະໄຟລມີຂໍ້ມູນເປັນແບບໃດ

ແລະ ໃນການ Commit ແຕ່ລະເທື່ອຈະຕ້ອງໃສ່ Commit Message ເພື່ອອະທິບາຍລາຍລະອຽດຂອງຂໍ້ມູນໃນ Commit ນັ້ນໆວ່າເຮົາເຮັດຫຍັງໄປແດ່ ເພື່ອມາເບິ່ງໃນພາຍຫຼັງຈະໄດ້ອ່ານຈາກ Commit Message ໄດ້ເລີຍ ບໍ່ຕ້ອງໄປນັ່ງກົດເບິ່ງເທື່ອລະໄຟລ ດັ່ງນັ້ນການ Commit ທີ່ດີຈຶ່ງຄວນໃສ່ໃຈກັບ Commit Message ນຳອີກ



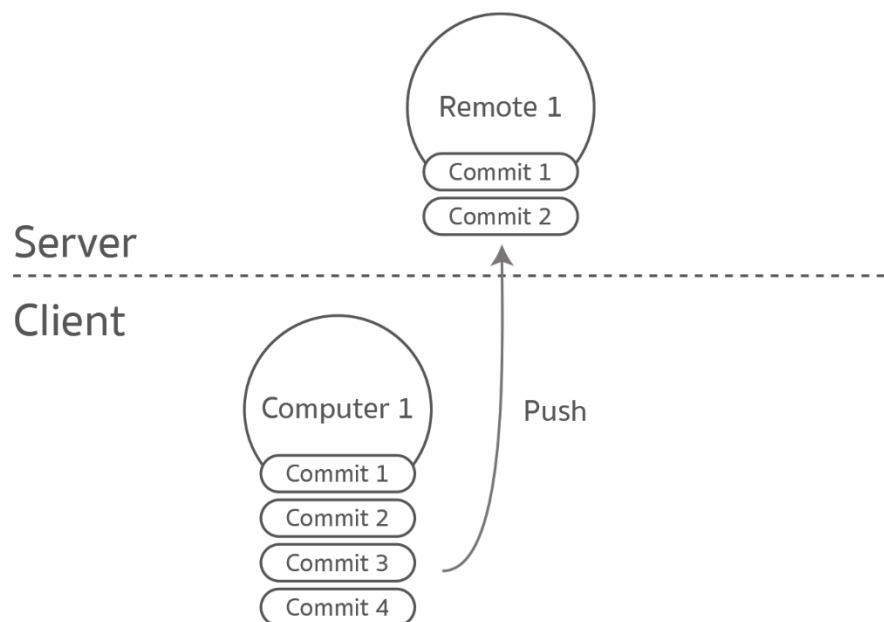
## ງ. Unstaged ແລະ Staged

ຂໍເວົ້າເຖິງ 2 ຄຳນີ້ໄປພ້ອມໆກັນເລີຍ

ເວລາເຮົາແກ້ໄຂໂຄດ ຫຼື ແກ້ໄຂຂໍ້ມູນ ໄຟລທີ່ຖືກແກ້ໄຂຈະຢູ່ໃນສະຖານະ *Unstaged* ແລະ ເວລາທີ່ເຮົາເຮັດຫຍັງສຳເລັດຮຽບຮ້ອຍແລ້ວຢາກຈະ Commit ເກັບໄວ້ ຈະຕ້ອງເລືອກໄຟລທີ່ຕ້ອງການເພື່ອຍ້າຍເຂົ້າສູ່ໃນສະຖານະ *Staged* ກ່ອນເຖິງຈະກໍານົດ Commit ໄດ້, ຊຶ່ງສະຖານະ *Unstaged* ແລະ *Staged* ເຮັດໃຫ້ເຮົາສາມາດເລືອກສະເພາະບາງໄຟລສຳລັບ Commit ໄດ້ນັ້ນເອງຈະໄດ້ Commit ສະເພາະໄຟລທີ່ເຮົາຊຽນສຳເລັດແລ້ວ

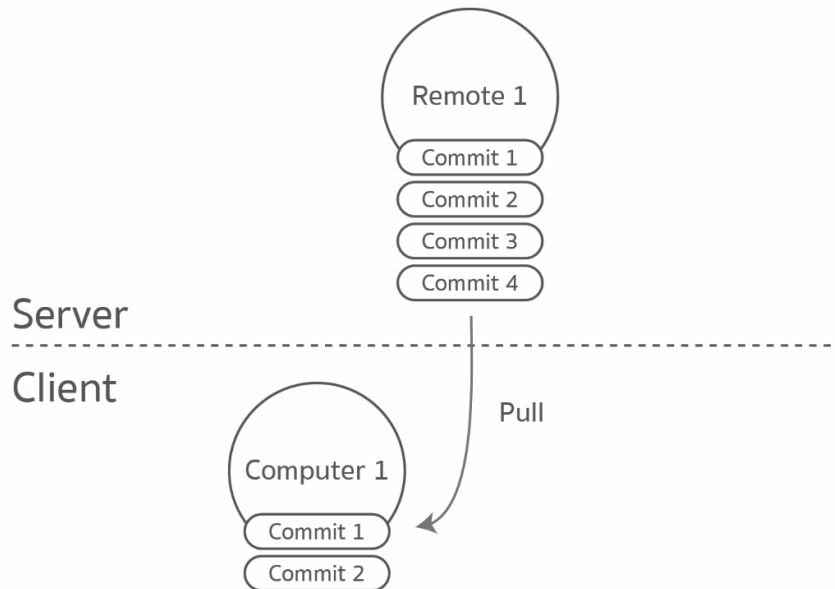
### ຈ. Push

ເວລາທີ່ມີ Commit ຢູ່ໃນເຄື່ອງ ແລະ ຕ້ອງການຈະ Sync ຂຶ້ນໄປເກັບໄວ້ໃນ Remote ຈະເອີ້ນຂັ້ນຕອນນີ້ວ່າ *Push*



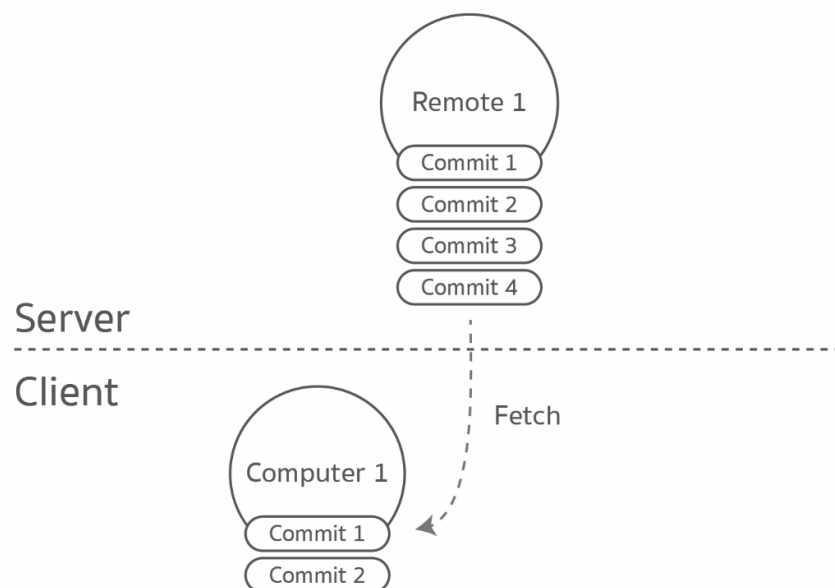
### ສ. Pull

ເວລາ Sync ຈາກ Remote ເພື່ອດຶງຂໍ້ມູນ Commit ໃໝ່ໆລົງມາເກັບໄວ້ໃນເຄື່ອງຈະເອີ້ນຂັ້ນຕອນນີ້ວ່າ *Pull*



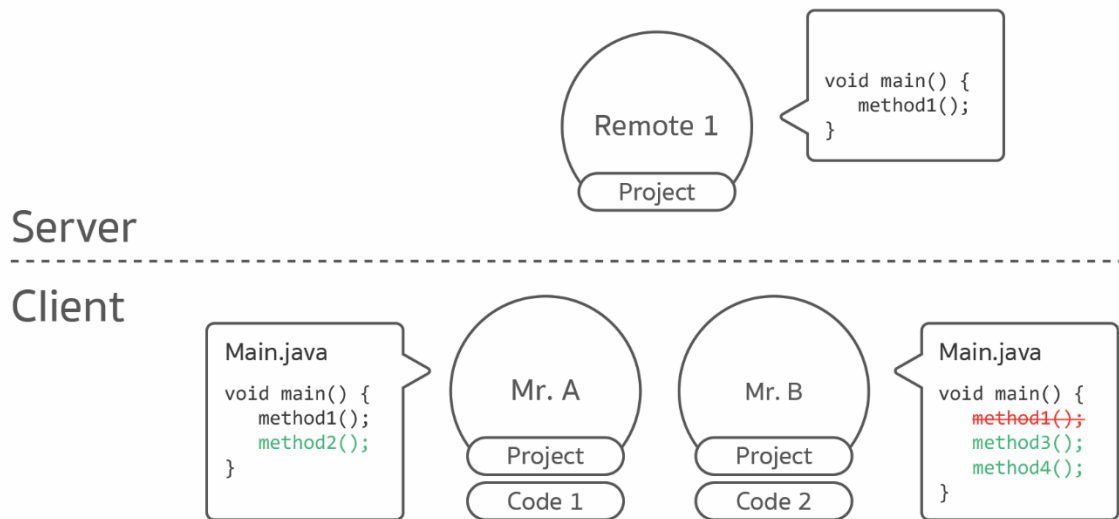
#### ຊ. Fetch

ໃນບາງເທື່ອເຮົາອາດຈະບໍ່ຕ້ອງການ Pull ຂໍ້ມູນລົງມາເກັບໄວ້ໃນເຄື່ອງທັນທີ ແຕ່ຢາກເຊັກສະຖານະຂອງ Remote ຊຶ່ງວ່າມີໃຜ Push ຂໍ້ມູນໃໝ່ຂຶ້ນໄປທີ່ Remote ຫຼື ບໍ່ ເຮົາເອີ້ນວິທີນີ້ວ່າ *Fetch* ຊຶ່ງການ Fetch ຈະເຮັດໃຫ້ເຮົາສາມາດອັບເດດ ແລະ ເບິ່ງ History ທັງໝົດທີ່ຢູ່ເທິງ Remote ໄດ້ໂດຍບໍ່ຕ້ອງດຶງຂໍ້ມູນລົງມາ. ຊຶ່ງທີ່ຈິງແລ້ວຂັ້ນຕອນການ Fetch ຈະຖືກເອີ້ນທຸກທີ່ເຮັດວຽກ Pull. ດັ່ງນັ້ນ ເຮົາຈຶ່ງສາມາດເລືອກໄດ້ວ່າຈະ Pull ເລີຍ (ມັນກໍຈະ Fetch ເພື່ອເຊັກເອງ) ຫຼື ຈະ Fetch ເບິ່ງກ່ອນວ່າມີ Commit ຫຍັງເພີ່ມເຂົ້າມາແລ້ວຄ່ອຍ Pull ລົງມາ

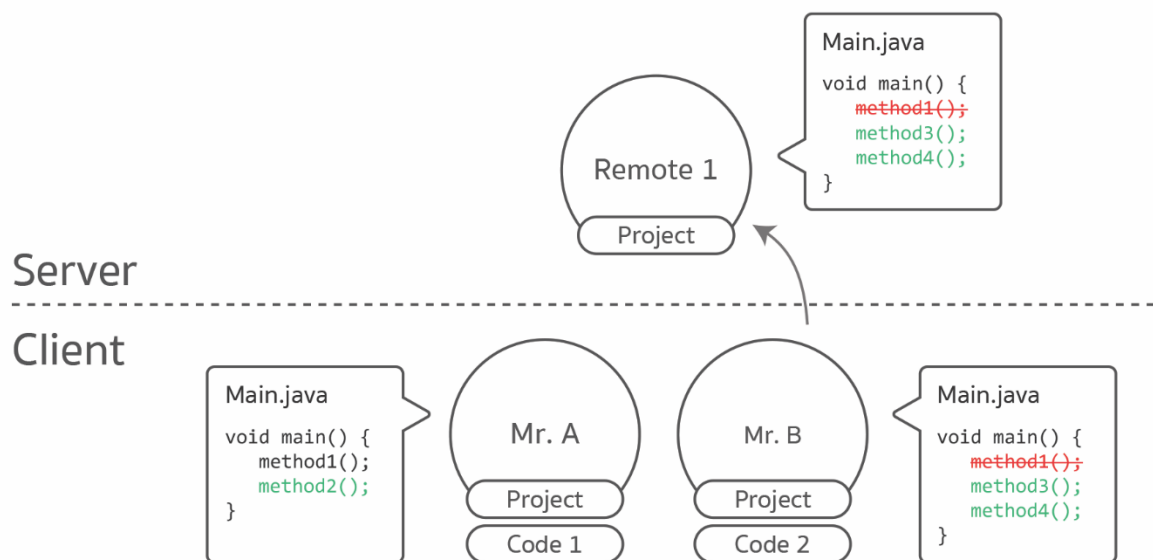


## ຍ. Merge Commit

ສົມມຸດວ່າ ທ້າວ A ກັບທ້າວ B ຂຽນໂຄ້ດນຳກັນຢູ່ ແລະ ທັງຄູ່ກໍຂຽນໂຄ້ດທີ່ຢູ່ໃນໄຟລດຽວກັນ

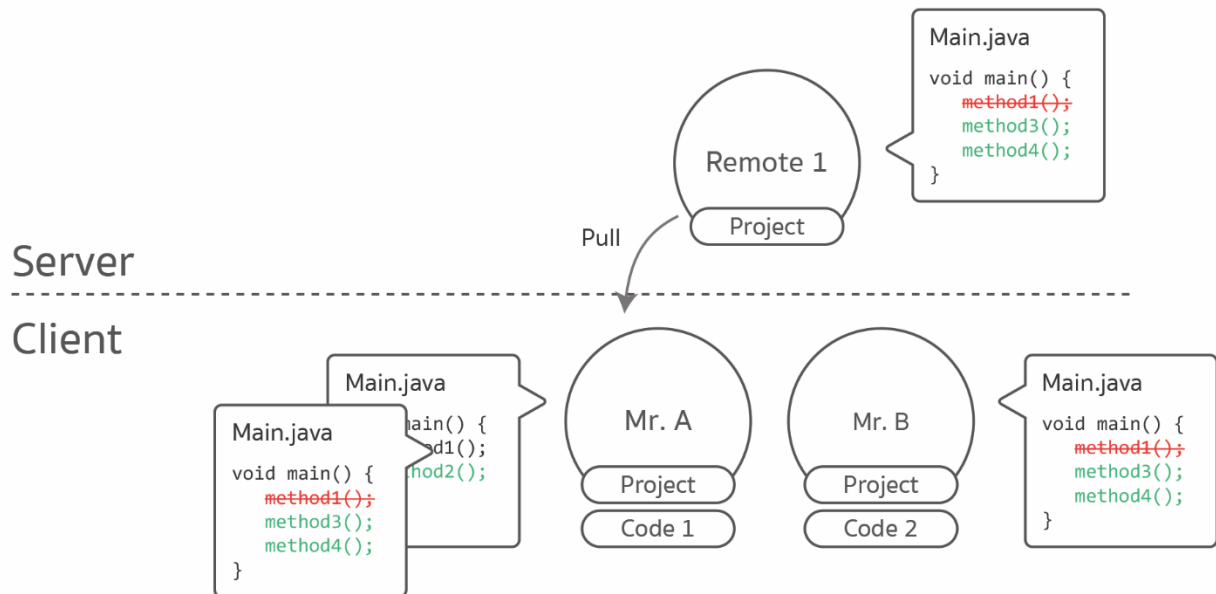


ທ້າວ B ຂຽນໂຄ້ດສຳເລັດແລ້ວ Commit ໄຟລທີ່ຕົວເອງຂຽນສຳເລັດແລ້ວ Push ຂຶ້ນ Remote



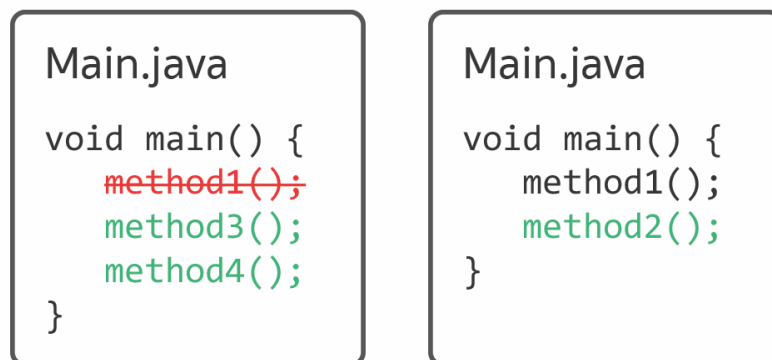
ທ້າວ A ຂຽນໂຄ້ດສຳເລັດພາຍຫລັງ ແລະ ຈະ Push ຂຶ້ນ Remote ແຕ່ພົບວ່າທ້າວ B ໄດ້ Push ຂຶ້ນໄປກ່ອນແລ້ວ ຈຶ່ງ Push ຂຶ້ນໄປທັນທີບໍ່ໄດ້

ດັ່ງນັ້ນສິ່ງທີ່ທ້າວ A ຕ້ອງເຮັດກ່ອນທີ່ຈະ Push ຂອງຕົວເອງຂຶ້ນໄປໄດ້ກໍຄືຈະຕ້ອງ Pull ຈາກ Remote ລົງມາໃໝ່ກ່ອນເພື່ອອັບເດດ Commit ທີ່ທ້າວ B ໄດ້ Push ຂຶ້ນໄປ ຊຶ່ງເຮົາເອີ້ນຂັ້ນຕອນນີ້ວ່າ *Merge Commit* ນັ້ນເອງ



#### ໓. Conflict

ໃນຂະນະທີ່ກຳລັງຈະ Merge ຢູ່ນັ້ນຈະເຫັນວ່າໂຄ້ດຂອງທ້າວ A ແລະ ທ້າວ B ມັນຕຳກັນໄປ ແກ້ໂຄດຍ່ອນດຽວກັນ ດັ່ງນັ້ນ Git ຈະແຈ້ງວ່າເກີດ *Conflict* ຫຼື ກໍຄືໂຄດທີ່ທັບຊ້ອນກັນນັ້ນເອງ



ຊຶ່ງທ້າວ A ກໍຕ້ອງແກ້ Conflict ນີ້ໃຫ້ຮຽບຮ້ອຍເຖິງຈະ Merge Commit ແລ້ວ Push ຂຶ້ນ Remote ໄດ້ ຊຶ່ງຕອນທີ່ເກີດ Conflict ຂຶ້ນແລ້ວເປີດໂຄ້ດດັ່ງກ່າວມາເບິ່ງ ກໍຈະເຫັນເປັນແບບນີ້

```

void main() {
<<<<<<< HEAD
    method1();
    method2();
=====
    method3();
    method4();
>>>>>>> 322d39a003e4d9...
}

```

ໃຫ້ສັງເກດຮູບແບບຂອງເຄື່ອງໝາຍ <<<, === ແລະ >>> ໃຫ້ດີໆ ແລ້ວຈະເຫັນວ່າທີ່ຈິງແລ້ວ ມັນມີຮູບແບບທີ່ເຂົ້າໃຈໄດ້ງ່າຍໆໂດຍທີ່

- ໂຄດທີ່ຢູ່ລະຫວ່າງ <<< ແລະ === ຄືໂຄດຂອງທ້າວ A
- ໂຄດທີ່ຢູ່ລະຫວ່າງ === ແລະ >>> ຄືໂຄດຂອງທ້າວ B ສ່ວນຕົວເລກຕໍ່ທ້າຍຄືໝາຍເລກຂອງ Commit ບ່ອນເຮັດວຽກ Merge

ເພື່ອໃຫ້ໂຄດເຮັດວຽກໄດ້ເໝາະສົມ ທ້າວ A ກໍຈະຕ້ອງເອົາໂຄດທີ່ຕົວເອງແກ້ໄຂໄປທັງໝົດກັບໂຄດຂອງທ້າວ B ໃຫ້ເຮັດວຽກໄດ້

```

void main() {
    method2();
    method3();
    method4();
}

```

ເມື່ອສໍາເລັດແລ້ວກໍໃຫ້ Commit (ຊຶ່ງຈະເປັນການ Merge Commit) ແລ້ວທ້າວ A ກໍຈະສາມາດ Push ຂຶ້ນ Remote ໄດ້ແລ້ວ (ທ້າວ B ກໍຈະຕ້ອງ Pull Commit ພາຍຫລັງ ແລະ ຖ້າຫຼົງໄປແກ້ໄຂຊຳກໍອາດຈະເກີດ Conflict ໄດ້)

Conflict ຖືວ່າເປັນເຫດການທີ່ເກີດຂຶ້ນໄດ້ເປັນປົກກະຕິ ໂດຍສະເພາະຢ່າງຍິ່ງໂປຣເຈັກໃຫຍ່ໆ ທີ່ມີ Developer ຫຼາຍໆຄົນຊ່ວຍກັນຂຽນໂຄດ ດັ່ງນັ້ນ, ການແກ້ Conflict ຈຶ່ງເປັນໜຶ່ງໃນພື້ນຖານຂອງການໃຊ້ງານ Git ທີ່ນັກພັດທະນາຕ້ອງເຂົ້າໃຈ ແລະ ຈັດການກັບມັນບໍ່ໄດ້ ສະນັ້ນ ຈະເກີດບັນຫາຢ່າງເຊັ່ນ ຫຼົງໄປລົບໂຄດຂອງໝູ່ໂດຍບໍ່ສົນໃຈຫຍັງເພື່ອໃຫ້ Conflict ຫາຍໄປເປັນຕົ້ນ

➤ ເມື່ອສະຫລຸບຂັ້ນຕອນຂອງການແກ້ Conflict ຈະເປັນແບບນີ້

- Pull ຈາກ Remote ລົງເຄື່ອງ
- Conflict ເກີດຂຶ້ນ
- ແກ້ໄຂໂຄດໃຫ້ເໝາະສົມ

- Merge Commit
- Push ຂຶ້ນ Remote

➤ ສະຫຼຸບຮອບວຽນຂອງ Git ໃນຊຸດທຳອິດ

❖ ເວລາພັດທະນາໂປຣແກຣມກໍຈະມີຂັ້ນຕອນແບບນີ້

- ຂຽນໂຄດ
- ເລືອກໄຟລທີ່ຕ້ອງການເຂົ້າ Staged
- Commit
- ຂຽນໂຄດຕໍ່
- ເລືອກໄຟລທີ່ຕ້ອງການເຂົ້າ Staged
- Commit
- ວິນຫຼຸບໄປເລື້ອຍໆຈົນກວ່າງານຈະສຳເລັດ

❖ ເວລາຈະ Sync ຂໍ້ມູນໄປທີ່ Remote ກໍຈະມີຂັ້ນຕອນແບບນີ້

- Fetch ເພື່ອເຊັກວ່າມີໃຜ Push ຫຍັງຂຶ້ນ Remote ຫຼື ບໍ່ (ບໍ່ຈຳເປັນຕ້ອງເຮັດກໍໄດ້)
- ຖ້າມີກໍຈະຕ້ອງ Pull ໂຄດກ່ອນແລ້ວ Merge Commit ໃຫ້ຮຽບຮ້ອຍ
- Push ຂໍ້ມູນຂຶ້ນ Remote

❖ ເວລາຈະ Sync ຂໍ້ມູນໄປທີ່ Remote ແບບມີ Conflict ກໍຈະມີຂັ້ນຕອນແບບນີ້

- Fetch ເພື່ອເຊັກວ່າມີໃຜ Push ຫຍັງຂຶ້ນ Remote ຫຼື ບໍ່ (ບໍ່ຈຳເປັນຕ້ອງເຮັດກໍໄດ້)
- Pull ເພື່ອດຶງຂໍ້ມູນມາໄວ້ໃນເຄື່ອງ
- ເກີດ Conflict
- ແກ້ໄຂໂຄດໃຫ້ເໝາະສົມ
- Merge Commit
- Push ຂໍ້ມູນຂຶ້ນ Remote

❖ ຂຽນໂຄດໄປຈົນເຖິງເມື່ອໃດເຖິງຈະຕ້ອງ Commit?

ອັນນີ້ໜ້າຈະເປັນຄຳຖາມທີ່ເກີດຂຶ້ນເລື້ອຍກັບນັກພັດທະນາມືໃໝ່ທີ່ກຳລັງເລີ່ມຕົ້ນຮຽນຮູ້ ແລະ ຫັດໃຊ້ງານ Git ກັບໂປຣເຈັກຂອງຕົວເອງ, ໃນການ Commit ແຕ່ລະເທື່ອເຖິງແມ່ນວ່າຈະບໍ່ໄດ້ມີໃຜມາກຳນົດມາດຕະຖານ ແຕ່ການ Commit ທີ່ດີຄວນເຮັດໃຫ້ສາມາດໄລ່ໂຄດພາຍຫຼັງໄດ້ງ່າຍສາມາດເຮັດວຽກຮ່ວມກັບຄົນອື່ນໄດ້ ແລະ ຕ້ອງເຮັດຄືເກົ່າທຸກເທື່ອຈົນເປັນນິໄສ ບໍ່ແມ່ນວ່າຄິດອອກເມື່ອໃດກໍຄ່ອຍ Commit

ສຳລັບການ Commit ທີ່ດີຕ້ອງກະຊັບ ແລະ ໜ້ອຍຫຼາຍພໍທີ່ຈະໄລ່ເບິ່ງໃນ History ໄດ້ງ່າຍໆ ເວລາມີໂປຣເຈັກໜຶ່ງ ຈະຕ້ອງແຍກສິ່ງທີ່ຕ້ອງເຮັດອອກມາເປັນ Feature ແລ້ວກໍແບ່ງຍ່ອຍລົງໄປອີກໃຫ້ເປັນລະດັບ Function ຈຶ່ງຈະເລີ່ມຂຽນໂຄດ.

ເມື່ອເຮັດໃນແຕ່ລະ Function ສໍາເລັດກໍຈະ Commit ໃຫ້ຮຽບຮ້ອຍແລ້ວກໍເຮັດ Function ຕໍ່ໄປ ແລະ ເຮັດວິນແບບນີ້ໄປເລື້ອຍໆຈົນ Feature ນັ້ນສໍາເລັດ.

ບໍ່ແນະນຳໃຫ້ເຮັດໃນລະດັບ Feature ສໍາເລັດແລ້ວຄ່ອຍ Commit ເພາະການເຮັດແບບນັ້ນຈະໃຊ້ປະໂຫຍດ Git ໄດ້ແຕ່ການ Backup ເທົ່ານັ້ນ ແຕ່ຈະໄລ່ເບິ່ງ History ເວລາແກ້ໄຂໂຄດໄດ້ຍາກ ແລະ ບໍ່ສາມາດຍ້ອນຂໍ້ມູນກັບໄດ້ເລີຍ ເພາະມັນຈະຍ້ອນກັບໄປທັງໝົດເລີຍ ໂຄດທີ່ຂຽນໄວ້ທັງໝົດກໍຈະຫາຍໄປພ້ອມໆກັນ.

ແລະ ກໍບໍ່ແນະນຳໃຫ້ Commit ໂຄດທີ່ຍັງຂຽນບໍ່ສໍາເລັດເຊັ່ນກັນ ໂຄດທີ່ຂຽນບໍ່ສໍາເລັດ ໝາຍເຖິງໂຄດທີ່ບໍ່ສາມາດກົດ Run ເພື່ອທົດສອບໂປຣແກຣມໄດ້ ເພາະຖ້າເຮັດວຽກຮ່ວມກັບຄົນອື່ນແລ້ວຄົນອື່ນ Pull ຂໍ້ມູນໄປ ກາຍເປັນວ່າຄົນໆນັ້ນກໍ Run ໂປຣແກຣມບໍ່ໄດ້ເຊັ່ນກັນ ຕ້ອງເສຍເວລານຶ່ງແກ້ໂຄດຂອງເຮົາອີກ ແລະ ຢ່າລືມໃສ່ໃຈກັບ Commit Message ທ້າມໃສ່ຫຍັງແບບນີ້ໂດຍເດັດຂາດ ເພາະມັນຈະບໍ່ມີທາງຮູ້ເລີຍວ່າ Commit ນັ້ນເຮັດຫຍັງລົງໄປ.

- “Edit code”
- “Commit”
- “Refactor”
- ❖ ຕ້ອງ Push ເມື່ອໃດ?

ນອກເໜືອຈາກການ Commit ແລ້ວການ Push ກໍເປັນອີກຢ່າງທີ່ຫຼາຍໆຄົນສົງໄສເຊັ່ນກັນ ເນື່ອງຈາກການ Push ນັ້ນໃນບາງເທື່ອກໍຕ້ອງ Pull ແລ້ວ Merge Commit ໃຫ້ຮຽບຮ້ອຍກ່ອນເຖິງຈະເຮັດໄດ້ ດັ່ງນັ້ນ ບໍ່ແນະນຳໃຫ້ Push ກ່ອນປິດຄອມ ເນື່ອງຈາກຖ້າມີຄົນ Push ກ່ອນໜ້າ ແລ້ວພໍເຈົ້າ Pull ມາກໍພົບວ່າມັນ Conflict ສະນັ້ນ ສຸດທ້າຍແລ້ວເຈົ້າກໍຕ້ອງນຶ່ງແກ້ Conflict ຊຸດໃຫຍ່ ດັ່ງນັ້ນທາງທີ່ດີຄວນ Push ທຸກເທື່ອທີ່ມີໂອກາດດີກວ່າ ຢ່າງນ້ອຍຖ້າມັນຈະ Conflict ກໍໄດ້ແກ້ຕັ້ງແຕ່ເນື້ອງ ແລະ ບໍ່ຫຼາຍປານໃດ ທີ່ສໍາຄັນມັນຈະຊ່ວຍໃຫ້ທຸກຄົນໃນທີມຮູ້ Progress ນຳກັນໄດ້ຈາກສະຖານະການ Push ອີກດ້ວຍ ແລະ ນອກຈາກນີ້ ໃນບາງບ່ອນທີ່ມີການເຮັດ Continuous Integration (CI) ທີ່ຈະດຶງ Commit ຈາກ Remote ໄປເຮັດ Testing ຖ້າເຮົາ Push ເລື້ອຍໆແລ້ວ Test ເກີດ Failed ຂຶ້ນມາກໍຈະເຮັດໃຫ້ຮູ້ຕົວໄດ້ໄວ ແລະ ແກ້ໄຂໄດ້ທັນທີ (<https://blog.nextzy.me/ມາເຮື້ນຮູ້-git-ແນວນ່າຍາກກັນເລື້ອຍ-427398e62f82> 16 Aug 2017).

#### 10.1.3.7. GraphQL API

GraphQL ຄືພາສາສໍາລັບການເຂົ້າເຖິງຂໍ້ມູນ (Query Language) ເພື່ອການໃຊ້ງານ API ຂອງລະບົບ ແລະ ຈະປະມວນຜົນຄໍາສັ່ງທີ່ຝັງ server ຫຼື ທີ່ເອີ້ນວ່າ server-side runtime ໂດຍໃຊ້

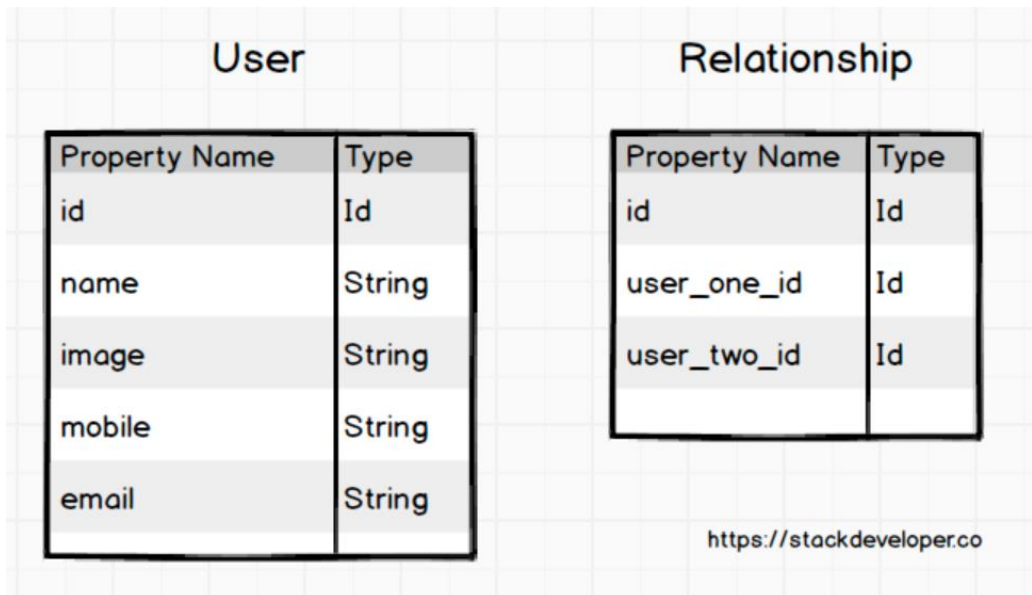
ໂຄງສ້າງຂໍ້ມູນທີ່ເຮົາກຳນົດໄວ້. ແຕ່ GraphQL ບໍ່ໄດ້ພັດທະນາຂຶ້ນມາເພື່ອແທນທີ່ພາສາສຳລັບການເຂົ້າເຖິງຂໍ້ມູນເຊັ່ນ SQL ຫຼື ເຮັດໜ້າທີ່ເປັນລະບົບຈັດເກັບຖານຂໍ້ມູນ (storage engine) ແຕ່ຢ່າງໃດ

ໃນປະຈຸບັນວິທີທີ່ເຂົ້າເຖິງຂໍ້ມູນຜ່ານເວັບເຊີວິດທີ່ໄດ້ຮັບຄວາມນິຍົມກໍຈະເປັນ REST API ຜ່ານທາງ HTTP Methods ຕົວຢ່າງເຊັ່ນ ຕ້ອງການດຶງຂໍ້ມູນຜູ້ໃຊ້ງານທັງໝົດ ກໍສາມາດຮຽກຜ່ານ API ໄດ້ດັ່ງນີ້

```
GET /users
      ຫຼື ຫາກຕ້ອງການດຶງເພື່ອນທັງໝົດຂອງຜູ້ໃຊ້ງານ ID 25
GET /users/25/friends
```

ແຕ່ຫາກຕ້ອງການດຶງເບີໂທສັບຂອງເພື່ອນທີ່ເປັນເພື່ອນກັບຜູ້ໃຊ້ງານ ID 25 ຈະຕ້ອງດຶງ ແລະ ຂຽນໂປຣແກຣມແບບໃດ? ໂດຍມີການຈັດເກັບຂໍ້ມູນດັ່ງນີ້

ຈາກຕົວຢ່າງນີ້ຈະເຫັນໄດ້ວ່າເຮົາຈຳເປັນຕ້ອງມີ API 2 end point ນຳກັນ ຄື GET /users/:id/friends ເພື່ອດຶງຂໍ້ມູນເພື່ອນຂອງຜູ້ໃຊ້ ID 25 ຈາກນັ້ນກໍວິນລູບດຶງຂໍ້ມູນຜູ້ໃຊ້ໂດຍໃຊ້ GET /users/:id ເພື່ອດຶງຂໍ້ມູນເບີໂທສັບອີກຄັ້ງ ແລະ ວິທີນີ້ກໍຈະໄດ້ຂໍ້ມູນອື່ນໆ ທີ່ບໍ່ຈຳເປັນຂອງຜູ້ໃຊ້ມາອີກດ້ວຍ ບໍ່ວ່າຈະເປັນຊື່, ຮູບພາບ ແລະ ອີເມວ



ໃນຄວາມເປັນຈິງແລ້ວ ອົງກອນໃຫຍ່ໆທີ່ມີຫຼາກຫຼາຍພະແນກ ຕ່າງກໍຮ້ອງຂໍຂໍ້ມູນທີ່ບໍ່ຄືກັນເຊັ່ນ ບາງພະແນກຕ້ອງການສະເພາະຂໍ້ມູນພະນັກງານ ບາງພະແນກຕ້ອງການຂໍ້ມູນພະນັກງານພ້ອມຂໍ້ມູນເງິນເດືອນ ຫຼື ບາງພະແນກຕ້ອງການຂໍ້ມູນພະນັກງານກັບຈຳນວນວັນທີ່ຂາດ ຫຼື ລາເທົ່ານັ້ນເປັນຕົ້ນ ການໃຊ້ງານ REST API ຈຶ່ງສ້າງຄວາມລຳບາກຕໍ່ການພັດທະນາ ແລະ ດູແລຮັກສາເຊີວິດ.



ດ້ວຍເທດນີ້ GraphQL ຈຶ່ງໄດ້ເຂົ້າມາເພື່ອແກ້ບັນຫາໃນຈຸດນີ້ໂດຍສະເພາະ ເພື່ອໃຫ້ເຮົາສາມາດດຶງຂໍ້ມູນໄດ້ກົງກັບຄວາມຕ້ອງການ ຫຼຸດຄວາມຊັບຊ້ອນໃນການຂຽນໂຄດເພື່ອດຶງຂໍ້ມູນ ສາມາດຈັດການ ແລະ ດູແລຮັກສາໂຄດໃນຝັ່ງ server-side ໄດ້ງ່າຍຫລາຍຢ່າງຂຶ້ນ ແລະ ທີ່ສຳຄັນ GraphQL ບໍ່ໄດ້ຜູກຕິດກັບ database ແລະ ທີ່ຈັດເກັບຂໍ້ມູນໃດໆທັງນັ້ນ

(ທຳຄວາມຮູ້ຈັກກັບ GraphQL [https://medium.com/@athivvat/ທຳຄວາມຮູ້ຈັກກັບ-graphql-980ac00d6c17#:~:text=GraphQL%20ຄື%20ພາສາສຳລັບການເຂົ້າສູ່ຂໍ້ມູນ%20\(Query%20Language\)%20ເພື່ອ,ຂໍ້ມູນທີ່ເຮົາກຳນົດໄວ້ 23 Feb 2020\).](https://medium.com/@athivvat/ທຳຄວາມຮູ້ຈັກກັບ-graphql-980ac00d6c17#:~:text=GraphQL%20ຄື%20ພາສາສຳລັບການເຂົ້າສູ່ຂໍ້ມູນ%20(Query%20Language)%20ເພື່ອ,ຂໍ້ມູນທີ່ເຮົາກຳນົດໄວ້ 23 Feb 2020).)

ວັນທີ ...../...../.....

ລາຍເຊັນຄະນະກຳມະການ

ວັນທີ ...../...../.....

ລາຍເຊັນອາຈານທີ່ປຶກສາ

ວັນທີ ...../...../.....

ລາຍເຊັນນັກສຶກສາ