مجلة جامعة تشرين للبحوث والدراسات العلمية _ سلسلة العلوم الهندسية المجلد (34) العدد (6) 2012
Tishreen University Journal for Research and Scientific Studies - Engineering Sciences Series Vol. (34) No. (6) 2012

إنقاص أزمنة الاستعلام ضمن قواعد البيانات اعتماداً على المسالك المتعددة للمعالجة

الدكتور جبر حنا*

(تاريخ الإيداع 1 / 7 / 2012. قُبِل للنشر في 13/ 11 / 2012)

□ ملخّص □

يعتمد عمل معظم البرامج على عمليات نقل البيانات من قواعد البيانات وإليها ، وتوجد هذه القواعد عادةً على مخدمات يجب عليها التعامل مع كم هائل من العمليات مثل عمليات الإدخال، والحذف، والاستعلام. وتتطلب هذه العمليات زمناً خاصاً لكل منها؛ وبهذا يكون معظم زمن التنفيذ المستهلك في البرامج هو زمن خاص بالعمليات على قواعد البيانات، وهذا الزمن هو جزء أساسي في أداء هذه البرمجيات.

حاول مصممو البرامج إنقاص هذا الزمن من خلال تطوير البرامج، وتحقيق أمثلية في تصميم قواعد البيانات، لكنهم لم يحصلوا على النتائج المرجوة من هذه العملية؛ لكون نظام التشغيل يفرض العديد من الشروط على هذه العملية؛ لذلك فإن استخدام المسالك المتعددة للمعالجة يعد حلاً جيداً في إنقاص أزمنة استعلامات قواعد البيانات.

الكلمات المفتاحية : مسارات متعددة – كتلة معالجة – قواعد بيانات – استعلام.

*أستاذ مساعد – قسم الحاسبات والتحكم الآلي – كلية الهندسة الميكانيكية والكهربائية – جامعة تشرين– اللاذقية –سورية.

مجلة جامعة تشرين للبحوث والدراسات العلمية _ سلسلة العلوم الهندسية المجلد (34) العدد (6) 2012
Tishreen University Journal for Research and Scientific Studies - Engineering Sciences Series Vol. (34) No. (6) 2012

Reducing of Data Base Query Time Using Multithreading

D. Jabr Hanna*

The work of most programs depends on data transfer to and from database ,and most of these database located on servers which have to deal with huge amount of insertion, deleting and queries operations. Each database operation needs its time; so, the database operation time is the most important time for program performance.

Program designers try to reduce this time by improving the code and optimizing database design, but these operations did not give the desired result because this operation is related at first to the operating system OS and multithreading or to the multicore of processor.

So, using multithreading in program design can reduce this time and improve the program performance.

Keywords : Multithreading – Process – Database – queries.

* **Associate Professor. Computer and Automatic Control, Faculty of Mechanical & Electrical Engineering, University of Tishreen**

مقدمة :

يعد زمن التنفيذ هو الجزء الأهم في تصميم البرمجيات؛ لذلك يعمل مصممو هذه البرمجيات على تقليل هذا الزمن بقدر الإمكان.

توجد العديد من المحددات والعوامل التي تؤثر في زمن تنفيذ البرمجيات، ويأتي في مقدمتها سرعة المعالج، ثم نظام التشغيل وأخيراً البرنامج؛ لذلك بدأ العمل على تطوير المعالجات من خلال زيادة سرعتها إلى أن وصلت هذه السرعة إلى حد أصبحت فيه هذه الزيادة على حساب استهلاك الطاقة وارتفاع حرارة المعالج؛ عندئذ بدأ العمل على تصميم معالجات متعددة النوى، حققت ارتقاء جيداً في أداء المعالجات، إضافة إلى القدرة على تنفيذ البرامج بالسرعة المطلوبة. وهنا أصبحت أنظمة التشغيل هي نقطة العثرة في عملية التطوير هذه؛ لهذا بدأ العمل على تطوير أنظمة التشغيل لتحاكي التطور الحاصل في بنية المعالجات، فظهرت الأنظمة ذات النظام 64 bit.

بعد هذا التطور في أنظمة التشغيل وبنية المعالجات، بقيت البرمجيات هي المشكلة الوحيدة التي باتت مخولة للاستفادة من هذه التطويرات لإنقاص زمن التنفيذ؛ أي يجب أن تحقق الاستفادة من هذه التطويرات في علميات الاستعلام من قواعد البيانات.

أهمية البحث وأهدافه :

من خلال هذا البحث يمكن أن نحقق النتائج الآتية :

- توضيح أهمية المسالك المتعددة في البرمجيات الحديثة.
- توضيح دور لغات البرمجة في تحقيق البرمجيات الحديثة ذات الكفاءة العالية.
- توضيح أهمية إنقاص الزمن المخصص لعمليات الاستعلام ضمن قواعد البيانات.
- تحديد آلية الاستفادة من المسالك المتعددة في تنفيذ العمليات على قواعد البيانات.
- تحديد معدل الإنقاص الممكن تحقيقه من خلال بناء الاستعلامات اعتماداً على المسالك المتعددة.

طرائق البحث ومواده :

العملية [1,4]:

بالتعريف هي برنامج قيد التنفيذ، وهذا البرنامج موجود بوصفه ملفاً على القرص الصلب في الصيغة المصدرية Source أو الشكل الثنائي Binary، لكن هذا الملف ليس بعملية؛ إذ يجب علينا أولاً استدعاء هذا البرنامج عن طريق الضغط على أيقونة البرنامج، ثم كتابة الاسم بوصفه أمراً، أو إذا كان البرنامج يبدأ عن طريق برنامج آخر فإن نظام التشغيل يجب أن ينشئ عملية لتشغيل البرنامج، فالعملية process هي نوع من الحاوي container للبرنامج، وهي مزودة من قبل نظام التشغيل، أما نظام التشغيل فسيحتاج إلى تزويد العملية عل الأقل بالعناصر الآتية :

ذاكرة memory : من أجل تحميل كود البرنامج ومعطياته.

1. مكدس stack : من أجل معالجة نداءات التوابع، والمتغيرات المحلية لتلك التوابع؛ أي يحوي مثلاً بارا متر subroutine، والمتغيرات المؤقتة، وعنوان العودة.
2. مراكم heap : من أجل تخصيص العناصر المنشأة بشكل ديناميكي.

3. بنى المعطيات data structures : من أجل إدارة العملية، مثل حفظ حالة الـ cpu عندما يتم تشغيل برامج أخرى، وكذلك تحتوي المعطيات المعرفة على أنها global.

المسار Thread[1,2,4,6]:

هو عبارة عن تدفق متسلسل وحيد للتحكم، والبرامج جميعها تحوي مساراً واحداً على الأقل، هو المسار الرئيسي، ويمكن أن ينفذ أكثر من مسار واحد في الوقت نفسه (التوازي الوهمي؛ أي يبدوان وكأنهما ينفذان في الوقت نفسه).

من الواضح أن المسالك مفيدة من أجل تحسين سرعة الحساب على المعالجات المتعددة. أما بالنسبة إلى الحواسيب ذات المعالج الوحيد، فإن الحسابات التي تتم عن طريق المسالك يمكن أن تتم بدونها.

مكونات العملية والمسار [1,2,4]:

كل مسار له مكوناته الخاصة وهي : عداد البرنامج، ومكدس التحكم (ينادي الإجرائيات ويعيدها)، ومكدس البيانات (المتحولات المحلية local variables).

ولكل المسالك مكونات مشتركة، وهي كود البرنامج، والمراكم، والصنف، ومتحولات الحالة.

مكونات كل من المسار والعملية مبينة في الجدول (1). ولابد من التنويه إلى أن مكونات العملية تكون مشتركة بين جميع المسالك داخل العملية، أما مكونات المسار فهي خاصة بكل مسار على حدا.

الجدول(1) مقارنة بين مكونات كل من العملية والمسلك

العناصر الموجودة ضمن المسار	العناصر الموجودة ضمن العملية
عداد البرنامج	فضاء العنونة
المسجلات	المتحولات العامة
المكدس	الملفات المفتوحة
الحالة	العمليات الأبناء

يمكن ملاحظة الاختلافات الآتية بين المسالك والعمليات :

- المسالك لها ذاكرة مشتركة بينما العمليات ليس لها ذاكرة مشتركة.
- المسالك أقل كلفة من حيث الإنشاء؛ ذلك أنها لا تحتاج إلى ذاكرة منسوخة.
- الاتصالات بين المسالك أسرع؛ وذلك بفضل الذاكرة المشتركة.
- عمليات الاتصال هذه يجب أن تنجز بشكل حذر؛ لتجنب تضاربات الذاكرة memory conflict، ومشاكل التوقيت timing problems.

الحاجة للمسارات [3,6,10]:

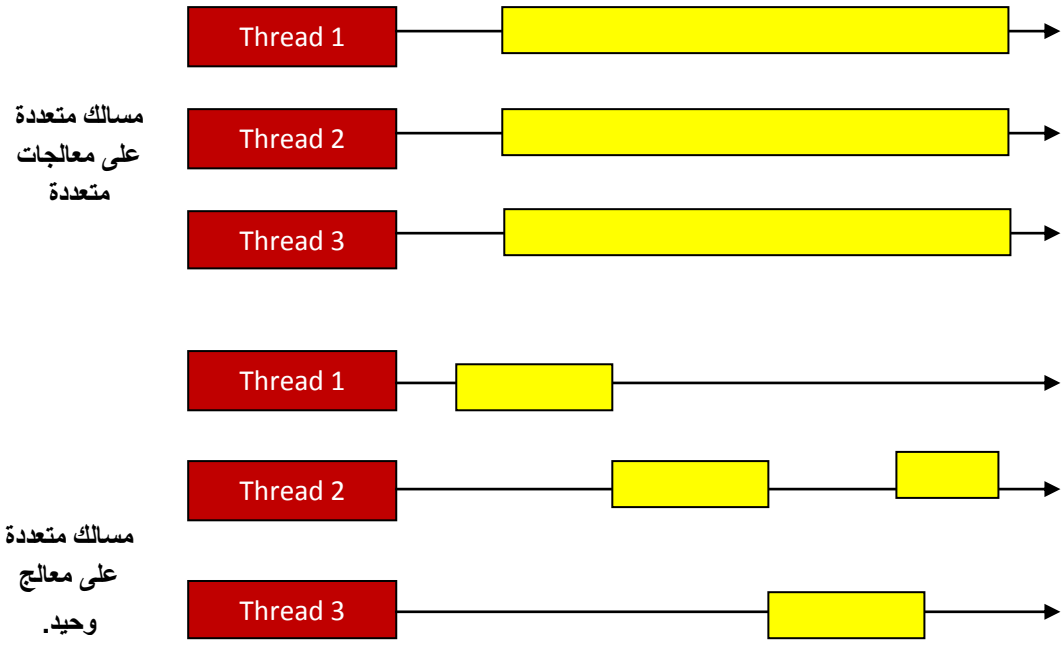
المسار – كما ذكرنا سابقاً– خفيف الوزن بالمقارنة مع العملية؛ ذلك انه يستخدم الموارد الأساسية؛ أي الذاكرة، ونظام التشغيل بشكل قليل، وهو أيضاً سريع الإنشاء والتلف.

والمسار هو الوحدة الأساسية لتخصيص العمل في نظام التشغيل، وتمثل هذه الوحدة وحدة الكود (مجموعة من التعليمات) التي يمكن أن تنفذ بواسطة المعالج cpu.

إذا المسار هو عملية خفيفة الوزن، وهو الأساس لكل العمليات ضمن وحدة المعالجة، وكل عملية تحوي مسلكاً واحداً على الأقل، وتتشارك هذه المسالك ضمن العملية الكود، والمراكم، والبيانات الموجودة فيها، وتتشارك المسالك التي تكون ضمن عملية ما قطاع الكود code section، وقطاع البيانات data section، وكذلك الموارد مثل الملفات المفتوحة الإشارات، والذاكرة المخصصة من الكومة عن طريق malloc.

ولكل مسار العناصر المميزة والفريدة الآتية: thread id، والمسجلات register، وفضاء المكدس stack space.

قد نستخدم المسالك المتعددة في بنية حاسوبية تحوي على عدة معالجات multiprocessor ، أو نستخدمها في بنية تحوي على معالج وحيد. ويوضح الشكل (1) الفرق بين استخدام المسالك المتعددة في كلتا الحالتين.



الشكل (1) الفرق بين استخدام المسالك المتعددة

وبهذا لابد لنا من التفريق بين التوازي والتزامن، فالتوازي هو مجموعة من الأنشطة التي تحدث في الوقت نفسه، مثل : أربع مهام يتم تنفيذها على أربع معالجات في الوقت نفسه. أما التزامن فهو القدرة على التشغيل في الوقت نفسه، مثل : أربع مهام يتم تقسيم الوقت بينها، وتنفذ على معالج واحد، ونطلق على هذا النوع التوازي الوهمي. فالتزامن إذاً يحوي حالة التوازي الحقيقي وحالة التوازي الوهمي (كلتا الحالتين). فالتزامن يتم فيه تنفيذ عمليتين أو أكثر في اللحظة الزمنية نفسها؛ وبذلك يكون هناك بيئة معالجة تفرعية حقيقية، وضمن هذه البيئة تتولى كل نواة عملية تقسيم العمل إلى مجموعة عمليات ومسالك، تقوم بتنفيذها على التوازي الوهمي.

نماذج برمجة المسالك [4,5] :

يوجد أربعة نماذج لبرمجة المسالك هي :

1. المعالجة الخفية أو اللاتوافقية :

تبدأ المسالك في هذه الآلية بمعالجة المهمة المستهلكة للوقت، في حين يستمر البرنامج الرئيسي في متابعة الأشياء الأخرى مع عدم الحاجة لانتظار المهمة المستهلكة للوقت لأن تكتمل. مثال : يستمر المستخدم باستخدام البرنامج عن طريق مسار واحد بينما يقوم المسار الآخر بإنجاز رتل قاعدة البيانات المستهلك للوقت Data Base Query.

2. المسار القائد :

يقوم أحد المسالك بلعب دور المسار القائد، ويستقبل جميع الطلبات بينما تبدأ بقية المسالك بمعالجة كل طلب، وهنا المسار القائد هو الوحيد القادر على تحديد المسار العامل، وتعد هذه المهمة هي المهمة الوحيدة له، فعلى سبيل المثال يقوم المسار المخدم الرئيسي في شبكة ما بالإصغاء إلى 80 منفذاً، بينما يبدأ المسار العامل الجديد بتنفيذ جميع طلبات Http.

3. أنبوبية المسالك :

يكون لدينا هنا باقة من المسالك، حيث يقوم كل مسلك بإنجاز عملية معالجة ما، ثم يقوم كل مسار بتمرير النتيجة إلى المسار التالي، ويتم ذلك على التسلسل، وهو في ذلك مشابه لشركة ما يقوم كل موظف فيها بإنجاز العمل المطلوب منه، حيث يمرر كل موظف هذا العمل إلى الموظف الذي يليه وهكذا..

4. المسالك الدوارة :

هذا النموذج هو آلية جديدة تمكننا من تجنب إنشاء مسار جديد لكل مهمة، ويمكن أن نستخدمه في حال المسار الرئيسي، حيث يكون لدينا باقة من المسالك التي تنشأ منذ البداية وتبقى في حالة انتظار، وحين يحتاج الماستر إلى القيام بعمل ما فإنه يقوم بالتقاط أحد المسالك الموجودة في حالة الانتظار، ثم يبدأ به. وميزات هذا النموذج أنه يقوم بتحسين السرعة.

فوائد المسالك :

1- الاستجابة : تسمح المسالك المتعددة للتطبيق أن يعمل حتى ولو كان جزء منه في حالة block.

2- مشاركة المواد : مشاركة الذاكرة، والملفات، وموارد أخرى للعملية التي تحوي على عدة مسارات موجودة ضمنها.

3- من الناحية الاقتصادية : إن إدارة العمليات وإنشاءها مكلف، ويستهلك وقتاً أكبر بكثير من إدارة المسالك وإنشائها.

4- استعمال مزايا المعالجات المتعددة : حيث إن كل مسار يمكنه أن يشغل وعلى التوازي؛ وذلك على معالج مختلف.

مساوئ المسالك :

1- استعمال ذروة العمليات الحسابية، حيث يكون لدينا ذروة من التزامن وإدارة المسالك.

2- ضبط البرمجة : من الصعب كتابة كود يحوي مسارات بشكل صحيح؛ لكون كل المسالك الموجودة ضمن العملية تشترك بالذاكرة ؛ لذا من الممكن لهذه المسالك أن تفسد بيانات عامة، وهذا يوجب على المبرمج أن يعالج حماية البيانات العامة كلها.

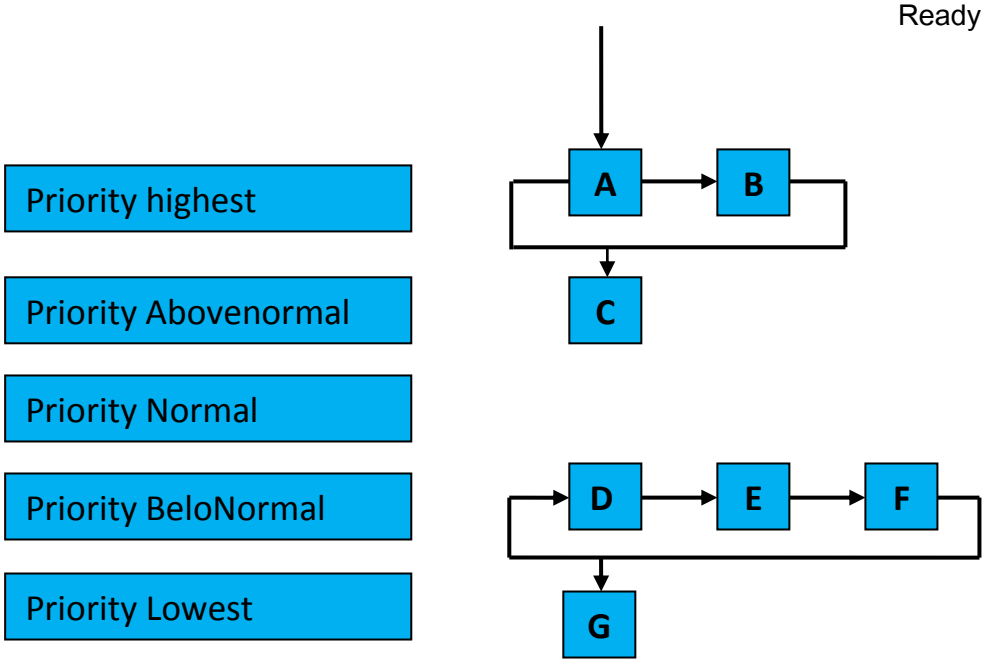
3-صعبة التنقيح : من الصعب تنقيح الكود الحاوي على مسارات؛ لأنه في بعض الأحيان قد نواجه مشاكل من الصعب جداً تعقبها.

أولويات المسالك [6] :

كل المسالك لها أولويات، وهذه الأولوية قد تكون : الأدنى، وعادية مائلة إلى الأدنى، وعادية، وعادية مائلة إلى الارتفاع، والأعلى، وتأخذ كل المسالك الأولوية العادية افتراضا default، ويمكن استخدام الخاصية priority لتغير أولوية المسار.

كذلك يعطي كل مسار فترة محددة من الوقت للتنفيذ، ونطلق على هذه الفترة الزمنية اسم quantum؛ وذلك قبل أن يعطي المعالج إلى مسار آخر، وسنلحظ بدون تقسيم الوقت أن المسالك تعمل لتنتهي قبل أن يقوم مسار آخر ببدء التنفيذ، وهذا ما نسميه تقسيم الوقت time- slicing.

كذلك يعمل مجدول نظام التشغيل على الإبقاء على المسار ذي الأولوية الأعلى قيد التنفيذ؛ وذلك في أي وقت كان من زمن تنفيذ البرنامج. وإذا وجدت عدة مسارات لها الأولوية نفسها، فإن هذا المجدول يقوم بالتنفيذ الحلقي لتلك المسالك، ولكن في بعض الأحيان قد تحدث المجاعة starvation فيقوم عندئذ المجدول بتأجيل تنفيذ المسار ذي الأولوية الأدنى والشكل(2) يوضح جدولة أولويات المسالك كما يلي:



الشكل (2) جدولة أولويات المسارات

وبهذا فإن المسار الذي يكون في حالة running، ويملك الأولوية الأعلى يكون فعالاً، ولكن في بعض الأحيان يحدث ما يسمى preemptivePriority-(إجهاض الأولوية)، ويحدث هذا عندما يستيقظ أحد المسالك ذي أولوية أعلى من المسار قيد التشغيل، وعندئذ يأخذ هذا المسار ذو الأولوية الأعلى التنفيذ بشكل أدنى. وينصح أن تعطى المسالك التي عليها أن تقوم بالكثير من العمل الأولوية الأدنى، أما المسالك المتعلقة بالدخل والخرج فيحبذ أن تعطى الأولوية الأعلى.

التطبيقات المعتمدة على المسالك المتعددة [6,7] :

يوجد نوعان من التطبيقات المعتمدة على المسالك المتعددة، وهي :

- التطبيقات التي يتم فيها جعل مجموعة من المسالك تقوم بمهام عدة وعلى التوازي، حيث لا يوجد أي اتصال بين المسالك مثال: GUI .
- التطبيقات التي يتم فيها جعل مجموعة من المسالك تقوم بمهام عديدة في الوقت نفسه؛ أي يوجد هنا اتصال بين المسالك مثال : data access .

لغات البرمجة والمسالك المتعددة :

تدعم معظم لغات البرمجة المسالك المتعددة للمعالجة، ولكن بإمكانيات متفاوتة، فلغة C++ بإصداراتها القديمة لا تدعم بشكل مباشر المسالك المتعددة للمعالجة، ولكن تزود إمكانية الوصول إلى بيئة المسالك المتعددة API الخاصة بنظام التشغيل.

أما اللغات الحديثة مثل Java,.net فقد حققت استخدام هذه المسالك بالنسبة إلى المطور ضمن البيئة نفسها.

كذلك عملت بعض اللغات على تحقيق بيئة التنفيذ المتوازي أو المسالك المتعددة مثل (OpenMP – MPI) ،وحاول بعضها الآخر العمل على تحقيق بنية تفرعية مثل (AtejiPY,CUDA).

لا تزال آليات التعامل مع المسالك المتعددة ضمن بيئات البرمجة صعبة من حيث الاستخدام، وخصوصاً في حال وجود بيانات مشتركة بين عدة مسارات، وقد يسبب هذا أخطاء في الحسابات أو أخطاء في زمن التنفيذ نتيجة محاولة الولوج المشترك من قبل أكثر من مسار، ولكن باتت هذه المسالك لا غنى عنها من أجل تحقيق تطبيقات الزمن الحقيقي.

النتائج والمناقشة :

صممنا في هذا البحث تطبيقاً ضمن بيئة net، يتولى عملية الدخول إلى قاعدة بيانات تحتوي على ثلاثة جداول لثلاثة صفوف، وكانت المهمة هي حساب معدل العلامات لكل صف، وتتطلب هذه العملية تنفيذ مجموعة من الاستعلامات ضمن كل جدول بحيث نستحصل علامات كل الطلاب ضمن الجدول لنحسب المعدل العام للصف.

تتكرر هذه العملية من أجل ثلاثة صفوف، وبذلك هناك كم هائل من الاستعلامات سنجريها لتحقيق هذا التطبيق.

اخترنا هذه المهمة لكونها تتطلب عمليات طويلة على الاستعلامات، وتؤمن رؤية جيدة عن دور المسالك المتعددة في إنقاص أزمنة الاستعلامات، ولتوضيح هذه الأهمية أجرينا عملية حساب المعدل بآليتين إحداهما تعتمد على مسلك معالجة وحيد، والأخرى تعتمد على المسالك المتعددة للمعالجة. ويوضح الشكل (3) واجهة المستخدم لهذا التطبيق.

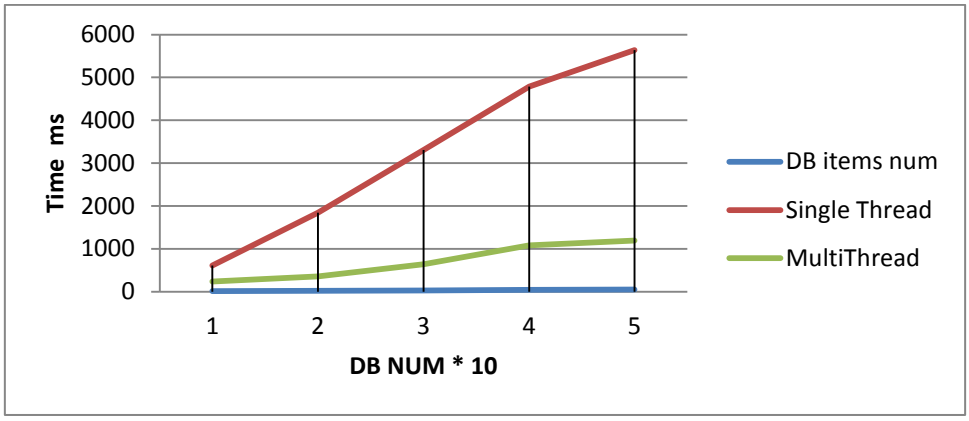
الشكل (3) واجهة المستخدم

بعد تجهيز التطبيق يجب إجراء الاختبارات على الأزمنة المستهلكة على الاستعلامات من أجل عدد متغير من العناصر ضمن كل جدول، وقد استخدمنا آليتين لتحقيق التطبيق الأولى تعتمد على مسلك معالجة وحيد يتم حساب كل العمليات ضمنه ، والثانية تتضمن ثلاثة مسالك للمعالجة يتم توزيع العمل بينها، حيث يتولى كل مسلك إجراء الحسابات الخاصة في كل صف. ونتائج هذا الاختبار موضحة في الجدول (2):

الجدول (2) نتائج المقارنة بين تعدد المسالك والمسلك الوحيد

Multithread VS Single thread DB time Compression		
DB items num	Single Thread	Multithread
10	612	238
20	1840	360
30	3311	642
40	4787	1081
50	5638	1192

ويمكن إظهار النتائج على شكل منحنٍ بياني بين الزمن المستهلك بتابعية لعدد العناصر ضمن الجدول، وكان المنحني كما في الشكل (4):



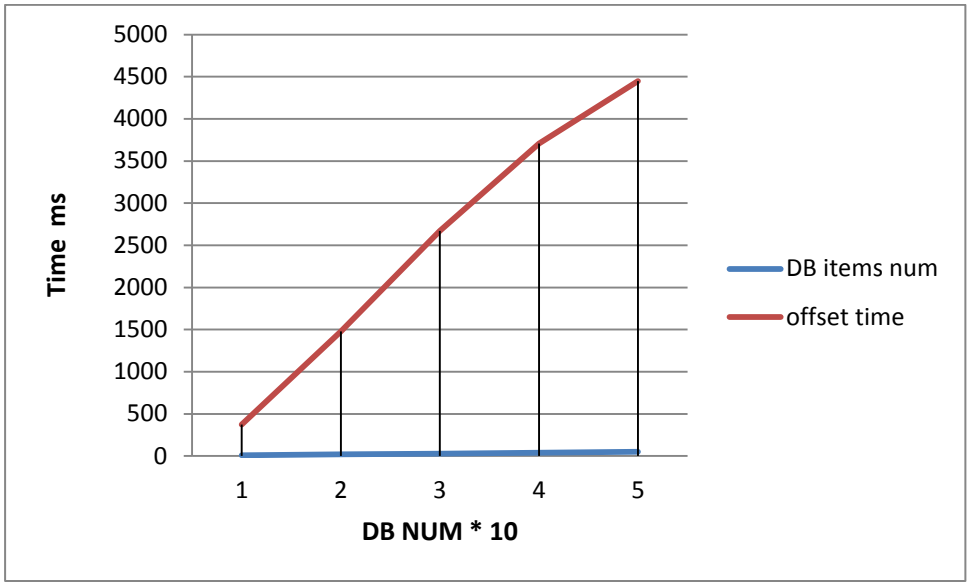
الشكل (4) الزمن المستهلك بوصفه تابعاً لعدد العناصر

يتجلى التطور الحاصل في الفارق الزمني بين العمليتين؛ أي بين الاستعلام من خلال مسار واحد والاستعلام من خلال مسارات متعددة؛ لذلك يجب حساب هذا الفارق وتسجيله في جدول كما يوضح الجدول (3):

الجدول (3) الفارق الزمني بين العمليتين

DB items num	deference time
10	374
20	1480
30	2669
40	3706
50	4446

وكذلك يجب رسم المنحني البياني المعبر عن تابع الفرق في التابعين السابقين كما في الشكل (5):



الشكل (5) الفرق الزمني بين التابعين

نلاحظ من خلال الشكل السابق أن العلاقة خطية لفرق الزمن بين العمليتين وعدد العناصر ضمن قاعدة البيانات؛ أي بمعنى آخر كلما زاد عدد العناصر ضمن قاعدة البيانات كان توفير الزمن أكبر.

الاستنتاجات والتوصيات :

من خلال هذا البحث نستطيع أن نستخلص النتائج الآتية :

- ✓ تساعد المسالك المتعددة في إنقاص زمن تنفيذ البرامج.
- ✓ يرتبط زمن الاستعلام ضمن أي جدول بعلاقة خطية مع عدد العناصر ضمن الجدول.
- ✓ يتزايد الفرق الزمني بين الاستعلام بمسار واحد والاستعلام من خلال المسالك المتعددة بتزايد حجم قاعدة البيانات.
- ✓ يؤثر عدد الجداول ضمن قاعدة البيانات بشكل مباشر في زمن الاستعلام.

المراجع:

1- Cameron Hughes, Tracey Hughes "*Object-oriented multithreading using C++"*Wiley Computer Pub., 1997 - Computers - 495.

2- Osni Marques, Michel Dayde, " *High Performance Computing for Computational Science",* Springer,470,2010.

3- Richard H.Carver, Kuo-Chung TAI," *Modern Multithreading Implementing, Testing And Debugging Multithreaded Java And C++/PThread/Win32 Program",* Wiley, 465 , 2006.

4- Edward L.Lamie, "Real-Time Embedded Multithreading Using Threadx And MIPS", Elsevier, 488, 2009.

5- Robert A.Lannucci, " *MultiThread Computer Architecture",* Kluwer Acadmic, 400, 1994.
6- Bil Lewis, Daniel J.Bery," MultiThreaded Programming With Java Technology", Sun, 495, 1997.
7- John Michael,"*Architecture Sensitive Data Base Query Processing on Chip Multiprocessors",*Columbia University,USA,2009.
8- Jingren Zhou, "*Improving Data Base Performance on Simultaneous Multithreading Processors",* ACMDL,2005.
9- Sahoo, "Query Slicing With Pipeline Processing", IEEE,2010.
10- Chowdhuri, "*Data Base System with Methodlogy for Parralel Schedule Generation in a Query Optimizer",* USA Patent,2009