لمحة عن علم تحليل البيانات

في إطار دراستنا للمقررات في السنوات السابقة ولمفاهيم تتعلق بنظم ادارة قواعد البيانات، ترد في أذهاننا العديد من التساؤلات عن علاقة هذه العلوم بعلوم حديثة أخرى ، ومنها علم تحليل البيانات.

ويعد علم تحليل البيانات علماً متطوراً ، وحيث يتعلق هذا العلم بمجالات متعددة، وأحد أدواته برنامج الاكسل، والأكسس، ولغة الاستعلامات الهيكلية SQL ، لذا كان لا بد لنا من أخذ لمحة بسيطة عن هذا العلم.

1-مقدمة:

في ظل عصرنا الغني بالبيانات، تشير دراسات أن ما لا يقل عن 2,5 كوينتيليون بايت من البيانات يتم إنتاجها يوميا، تعد قدرة المنظمات على استغلال البيانات بأفضل شكل ممكن؛ أحد مقومات النجاح الرئيسية لأي منظمة. ولكي تستفيد من تلك البيانات، يجب على المدراء أولا تحليلها.

هذه البيانات الضخمة في شكلها الأولي لا تعني أي شيء في الواقع لذا كان لا بد من تحليلها واستخراج المعلومات المفيدة منها وهنا يأتي دور علم البيانات الحديث الذي يعالج كل جزء من البيانات التي يتم إنشاؤها اليوم لتطوير وتيرة الأعمال واتخاذ القرارات الصائبة والموثوقة التي تعتمد على هذه المعلومات.

وتحليل البيانات جزء حيوي من إدارة أي عمل ناجح. وعندما تستخدم البيانات بشكل فعال، فإنها تضع المدراء على الطريق الصحيح نحو فهم أفضل لأداء لمنظماتهم وتحسين عملية اتخاذ القرارات فيما يتعلق بأنشطة المنظمة المستقبلية.

2-ما هو علم تحليل البيانات:

نستطيع تعريف تحليل البيانات أو الـ Data Analysis على أنها عملية ترتيب وتنقيح البيانات لاكتشاف معلومات مفيدة لاتخاذ قرارات خاصة بمجالات معينة مثل المال والأعمال، الصحة وغيرها، والغرض الأساسي من تحليل البيانات هو استخراج معلومات مفيدة منها واتخاذ قرارات مؤثرة بناءا عليها، أي أن محلل البيانات يستخدم علم تحليل البيانات في التعامل مع البيانات الضخمة لاستخراج معلومات مفيدة منها، وذلك بطلب من الشركات والمؤسسات الكبرى التي لديها هذا الكم من البيانات وتود تحليلها.

ويعد مجال تحليل البيانات كشكل من أشكال ذكاء الأعمال أو ما يعرف استخبارات الأعمال Business intelligence واختصاراBI ، وهو علم يستخدم لحل مشاكل وتحديات معينة داخل المؤسسات والشركات، وتكمن أهميته وقوته في العثور على مجموعات وأنماط البيانات التي يمكنها أن

تخبرنا بشيء مفيد وملائم حول مشكلة أو أمر معين من العمل يخص العملاء أو الموظفين أو المنتجات أو المخزون ...إلخ.

ولا يساعد هذا العلم على فهم السلوك الماضي فحسب بل يمكننا من التنبؤ بالاتجاهات والسلوكيات المستقبلية وبهذا تكون أي قرارات تتخذ مدروسة بناء على ما تخبرنا به البيانات وليست مجرد قرارات تعتمد على التخمين والحدس.

3-أهمية علم تحليل البيانات :

في عصر المعلومات، أصبحت البيانات هي الأصل الأكثر قيمة للشركات .من خلال تحويل هذه البيانات الخام إلى معلومات مفيدة، تستطيع الشركات فهم عملائها بشكل أفضل، واتخاذ قرارات مدروسة، وتحقيق نمو مستدام .فبدلا من أن تكون البيانات مجرد أرقام، يمكن تحويلها إلى رؤى قيمة تساعد الشركات على التنافس بفعالية في السوق.

هذه البيانات التي تجمع بكميات مهولة هي واحدة من أهم أصول الشركات التجارية وأكثرها استراتيجية في عالم الأعمال ، كما أن التنقيب في هذه البيانات وتحليلها وفهمها يساعدها على استخراج كنوز منها وهذه الكنوز هي معلومات قيمة تساعد أصحاب العمل على اتخاذ أفضل القرارات وتطوير الأداء بشكل مضمون.

ولا تقتصر فوائد تحليل البيانات على فهم سلوك العملاء فحسب، بل تمتد لتشمل تحسين العمليات التشغيلية، واكتشاف فرص جديدة، واتخاذ قرارات مدروسة .من خلال تحويل البيانات الخام إلى تقارير واضحة وموجزة، يمكن للشركات الاستفادة من هذه الرؤى القيمة لتحقيق أهدافها التجارية والنمو في سوق تنافسي.

4- أهم الجهات التي تعتمد على تحليل البيانات

- البنوك.
- شركات الاتصالات.
- شركات الأدوية.
- كبار المصنعين.
- الجامعات.
- مختبرات العلوم.
- منصات التواصل الاجتماعي.
- مواقع التجارة الإلكترونية.

5- أدوات تحليل البيانات الشائعة:

تتوفر مجموعة واسعة من الأدوات التي يمكن استخدامها لتحليل البيانات، ولكل أداة مزاياها وخصائصها التي تجعلها مناسبة لمجموعة متنوعة من المهام. إليك بعض الأدوات الشائعة المستخدمة في تحليل البيانات:

أ-لغات البرمجة:

- **Python :**تعتبر Python واحدة من أكثر لغات البرمجة شعبية لتحليل البيانات بفضل مكتباتها القوية ، وتتيح هذه المكتبات للمستخدمين تنفيذ مجموعة واسعة من المهام، من تنظيف البيانات وتصورها إلى بناء نماذج التعلم الآلي.
- **R:**لغة برمجة أخرى شائعة جدا في مجال الإحصاء وتحليل البيانات. تتميز R بمجموعة واسعة من الحزم التي تغطي جميع جوانب التحليل الإحصائي، مما يجعلها أداة قوية للباحثين والعلماء.

ب- أدوات التصور أو تصوير البيانات:

- **Tableau :**أداة قوية وسهلة الاستخدام لإنشاء لوحات معلومات تفاعلية ومرئيات جذابة. تسمح Tableauللمستخدمين بتوصيل البيانات من مصادر مختلفة وإنشاء تحليلات مرئية معقدة.
- **Power BI**: أداة أخرى قوية لإنشاء لوحات معلومات تفاعلية. تتكامل Power BI بشكل جيد مع منتجات Microsoft الأخرى، مما يجعلها خيارا جيدا للشركات التي تستخدم بالفعل منتجات Microsoft.

ج-قواعد البيانات:

- **SQL :**لغة الاستعلامات الهيكلية (Structured Query Language) هي لغة قياسية للتفاعل مع قواعد البيانات العلائقية. تستخدم SQL لاستخراج البيانات وتحديثها وتحليلها.
- **NoSQL** :مجموعة من قواعد البيانات غير العلائقية مصممة لتخزين أنواع مختلفة من البيانات غير المهيكلة وشبه المهيكلة، مثل البيانات النصية والبيانات المكانية.

د- أدوات أخرى:

- **Excel**:على الرغم من كونه تطبيقا لجدول البيانات، إلا أن Excel لا يزال يستخدم على نطاق واسع لتحليل البيانات البسيطة.
- **Hadoop:**منصة مفتوحة المصدر مصممة لتخزين ومعالجة كميات كبيرة من البيانات.
- **Spark**إ:طار عمل سريع لمعالجة البيانات الضخمة، يوفر أداء عاليا وتوسعا.

اختيار الأداة المناسبة يعتمد على عدة عوامل، بما في ذلك:

- **حجم البيانات :**للبيانات الضخمة، قد تحتاج إلى أدوات مثل Hadoop و.Spark
- **نوع البيانات :**البيانات المنظمة بشكل جيد يمكن تحليلها باستخدامSQL ، بينما البيانات غير المهيكلة قد تتطلب أدوات.NoSQL
- **المهام التحليلية :**إذا كنت تبحث عن بناء نماذج التعلم الآلي، فقد تحتاج إلى Python أو.R
- **المهارات الفنية :**مستوى مهارتك في البرمجة وتقنيات البيانات سيؤثر على اختيارك للأداة.
- **الميزانية :**بعض الأدوات مفتوحة المصدر، بينما البعض الآخر يتطلب ترخيصا.

6-تطبيقات علم تحليل البيانات

1-التسويق:

- **فهم سلوك العملاء :**يساعد تحليل البيانات في فهم سلوك العملاء وتفضيلاتهم، مما يتيح للشركات تطوير حملات تسويقية أكثر استهدافا وفعالية.
- **تخصيص العروض :**يمكن للشركات استخدام البيانات لتقديم عروض مخصصة لكل عميل، مما يزيد من احتمالية الشراء.
- **تحسين تجربة العملاء :**يساعد تحليل البيانات في تحديد نقاط الضعف في تجربة العملاء وتحسينها.
- **قياس أداء الحملات التسويقية :**يمكن للشركات قياس فعالية حملاتها التسويقية المختلفة وتحديد القنوات الأكثر فعالية.

.2-الرعاية الصحية:

- **اكتشاف الأمراض :**يساعد تحليل البيانات في اكتشاف الأمراض الجديدة وتطوير علاجات أكثر فعالية.
- **تحسين التشخيص :**يمكن استخدام البيانات لتحسين دقة التشخيص الطبي وتقديم رعاية مخصصة للمرضى.
- **إدارة المستشفيات :**يساعد تحليل البيانات في تحسين إدارة المستشفيات، مثل إدارة الموارد وتقليل وقت الانتظار.
- **تطوير الأدوية :**يمكن استخدام البيانات لتسريع عملية تطوير الأدوية الجديدة.

3-المالية:

- **تقييم المخاطر :**يساعد تحليل البيانات في تقييم المخاطر الاستثمارية واتخاذ قرارات استثمارية أكثر حكمة.
- **كشف الاحتيال :**يمكن استخدام البيانات لكشف الأنشطة الاحتيالية في المعاملات المالية.
- **توقع التغيرات في السوق :**يساعد تحليل البيانات في توقع التغيرات في السوق المالية واتخاذ قرارات استثمارية استباقية.

- **إدارة المحافظ الاستثمارية :**يمكن استخدام البيانات لتحسين إدارة المحافظ الاستثمارية وتحقيق عوائد أعلى.

4-مجالات أخرى:

- **الخدمات الحكومية :**تحسين تقديم الخدمات الحكومية، وتخطيط المدن، وإدارة الكوارث.
- **القطاع الصناعي :**تحسين عمليات الإنتاج، والتنبؤ بالصيانة، وتقليل التكاليف.
- **الرياضة :**تحليل أداء اللاعبين، وتطوير استراتيجيات اللعب، وتحسين التدريب.
- **التعليم :**تحسين تجربة التعلم، وتخصيص المناهج الدراسية، وتقييم أداء الطلاب.

باختصار، تحليل البيانات يمثل ثورة في العديد من المجالات، حيث يساعد في اتخاذ قرارات أفضل، وتحسين الكفاءة، واكتشاف فرص جديدة.

7-مستقبل علم تحليل البيانات:

لا شك أن علم تحليل البيانات يشهد تطورا متسارعا، مما يجعله أحد أهم العلوم في القرن الحادي والعشرين .مع تزايد حجم البيانات وتنوعها، يزداد الاعتماد على تحليل البيانات لاتخاذ القرارات الاستراتيجية في مختلف المجالات، ومنها:

- **الذكاء الاصطناعي وتعلم الآلة :**سيشهد اندماجا أكبر بين تحليل البيانات والذكاء الاصطناعي، مما يؤدي إلى ظهور نماذج أكثر تعقيدا ودقة في التنبؤ وتحليل البيانات.
- **تحليل البيانات بشكل سريع:** ستصبح القدرة على تحليل البيانات بشكل فوري أمرا حاسما في العديد من الصناعات، مما يتيح اتخاذ قرارات أسرع وأكثر استجابة.
- **أمن البيانات :**مع تزايد أهمية البيانات، ستزداد الحاجة إلى حماية البيانات وتأمينها من الاختراقات والهجمات السيبرانية.
- **تحليل البيانات في الحياة اليومية :**ستنتشر تطبيقات تحليل البيانات في جميع جوانب حياتنا، من الصحة والتعليم إلى النقل والتسوق.

الفصل الأول
المفاهيم الأساسية في نظم
إدارة قواعد البيانات

أولاً- المقدمة:

تلعب نظم إدارة قواعد البيانات (Data Base Management Systems) والمعروفة اختصاراً باسم (DBMS) دوراً مسيطراً في بناء نظم المعلومات الحديثة، حيث يتم تصميم وتشغيل معظم نظم المعلومات الحالية في المنظمات باستخدام نظم إدارة قواعد البيانات.

يعود السبب في ذلك إلى مجموعة من المزايا التي يؤمنها استخدامها نظم إدارة قواعد البيانات في تشغيل النظام المطور، والتي تعجز أساليب البرمجة التقليدية، المتمثلة باستخدام لغات البرمجة، عن تحقيقها، مثل: المرونة والاستقلالية والتكامل.

تشكل نظم إدارة قواعد البيانات أهم وأكثر تقنية مستخدمة في بناء نظم المعلومات وتشغيلها، يعود السبب في ذلك إلى مجموعة من المزايا التي يؤمنها استخدامها نظم إدارة قواعد البيانات في تشغيل النظام المطور، والتي يصعب تحقيقها في ظل البرمجة التقليدية، المتمثلة باستخدام لغات البرمجة، عن تحقيقها، مثل: المرونة والاستقلالية والتكامل.

ونظراً لأهمية هذه التقنية رأينا أن نقوم باستعراضها في إطار هذا الكتاب، لكي يستطيع القارئ تكوين فكرة عن آلية بناء النظام في ظل استخدام نظم إدارة قواعد البيانات.

من بين مختلف أشكال نظم إدارة قواعد البيانات تشكل نظم إدارة قواعد البيانات الترابطية (Relational Database Management Systems) أكثر الأنظمة استخداماً وانتشاراً في تطوير نظم المعلومات، لما تتمتع به النظم الترابطية من إمكانيات غير محدودة في تقويم مخزون البيانات وبساطة منطقها في عرض البيانات المخزنة بالنسبة للمستخدم.

يوجد في التداول مجموعة كبيرة من نظم إدارة قواعد البيانات الترابطية مثل (- Oracle Access-Ingers- FoxPro ..).

وقد رأينا أن نستخدم نظام (Microsoft Access) في هذا الكتاب لتوضيح كيفية بناء نظم المعلومات الإدارية في ظل قواعد البيانات الترابطية.

من المفيد في البداية التعريف بمكونات قاعدة البيانات، ثم التعرض بعد ذلك إلى النواحي الفنية والتقنية في قواعد البيانات من ناحية إنشاء قاعدة البيانات وإدخال البيانات إليها والحصول على المعلومات من قاعدة البيانات..الخ.

نظام قاعدة البيانات هو بشكل مبدئي حفظ السجلات بواسطة الحاسوب، وقاعدة البيانات بحد ذاتها هي عبارة عن مستودع إلكتروني لحفظ السجلات والملفات. ويمكن للمستخدم (User) تنفيذ عدد من العمليات على الملفات مثل:

١- إضافة ملف جديد إلى بنك المعلومات

٢- إضافة بيانات جديدة إلى الملفات الموجودة

٣- استرجاع البيانات من الملفات الموجودة

٤- تحديث البيانات في الملفات الموجودة

٥- حذف البيانات من الملفات الموجودة

٦- حذف بعض الملفات الموجودة من بنك المعلومات مثل الملفات الفارغة.

الشكل(١-١) يعرض لقاعدة بيانات صغيرة جداً تتكون من ملف واحد، يطلق عليه Customer تتضمن على البيانات المتعلقة بالعملاء في إحدى الشركات:

custNr	Name	Credit-Limit	City	TelNr
100	West Company	20000.00	Paris	5563215
110	Eastern Connection	14000.00	London	3952010
120	Inter Trade Company	5000.00	Amman	4412020
130	Home Service Company	8000.00	Damascus	2125683
140	LG Company	6000.00	Amman	4142050
150	Digital Company	12000.00	London	4785201

custNr	Name	Credit-Limit	City	TelNr
160	Modern Company	9000.00	Paris	9920142
170	Arabia Company	3000.00	Amman	5502147
180	Brada Company	7500.00	Damascus	3954280
190	Center City Company	10000.00	New York	1582014
200	Nora Company	15000.00	New York	8546377

الشكل (١-١) ملف العملاء

من خلال المثال المعروض في الشكل (١-١) يمكن ايضاح النقاط التالية:

١-لغرض التبسيط يطلق على الملفات المحوسبة مثل ملف العملاء اسم جدول وبدقة أكثر جداول ترابطية(Relational Tables).

٢- يمكن النظر إلى أسطر الجدول على أنها سجلات (Records) والأعمدة على أنها حقول (Fields) ضمن السجلات.

٣- نلاحظ أن بعض حقول الجدول تتضمن بيانات من النموذج الحرفي مثل أسم العميل والمدينة والبعض الأخر بيانات رقمية مثل مستوى الائتمان ورقم الهاتف.

٤-العمود رقم العميل يعد مفتاح رئيسي (Primary Key) لجدول العملاء أي أنه لايوجد عميلين لهما نفس الرقم.

٥- العمليات المبسطة التي تتم على قاعدة البيانات مثل Select, Update تتم من خلال استخدام لغة SQL وهي اختصار للكلمات (Structured Query Language) وهي عبارة عن لغة نمطية للتعامل مع قواعد البيانات الترابطية.

ثانياً- نظام قاعدة البيانات:

نظام إدارة قواعد البيانات هو عبارة عن حزمة برمجية جاهزة، تمكن المصمم من إنشاء قاعدة البيانات، وإجراء التعديلات عليها ومعالجة البيانات المخزنة في قاعدة البيانات من أجل الوصول إلى المعلومات المطلوبة من قبل المستخدمين.

يمكن النظر بشكل مبسط إلى قاعدة البيانات على أنها نظام لحفظ السجلات بواسطة الحاسوب أو أنظمة الملفات الإلكترونية، إنها عبارة عن مستودع لمجموعة من الملفات المخزنة في الحاسوب، ويزود المستخدم بمجموعة من التسهيلات من أجل تنفيذ عدد كبير ومتنوع من العمليات على هذه الملفات من بينها:

١- إضافة ملف جديد إلى قاعدة البيانات

٢- إضافة بيانات جديدة إلى الملفات الموجودة

٣- استرجاع البيانات من الملفات الموجودة

٤- تحديث البيانات في الملفات الموجودة

٥- حذف البيانات من الملفات الموجودة

٦- حذف بعض الملفات الموجودة من بنك المعلومات مثل الملفات الفارغة.

الغرض الرئيس لنظام قاعدة البيانات المحوسب هو تخزين البيانات والسماح للمستخدم باسترجاعها وتحديثها حسب حاجته.

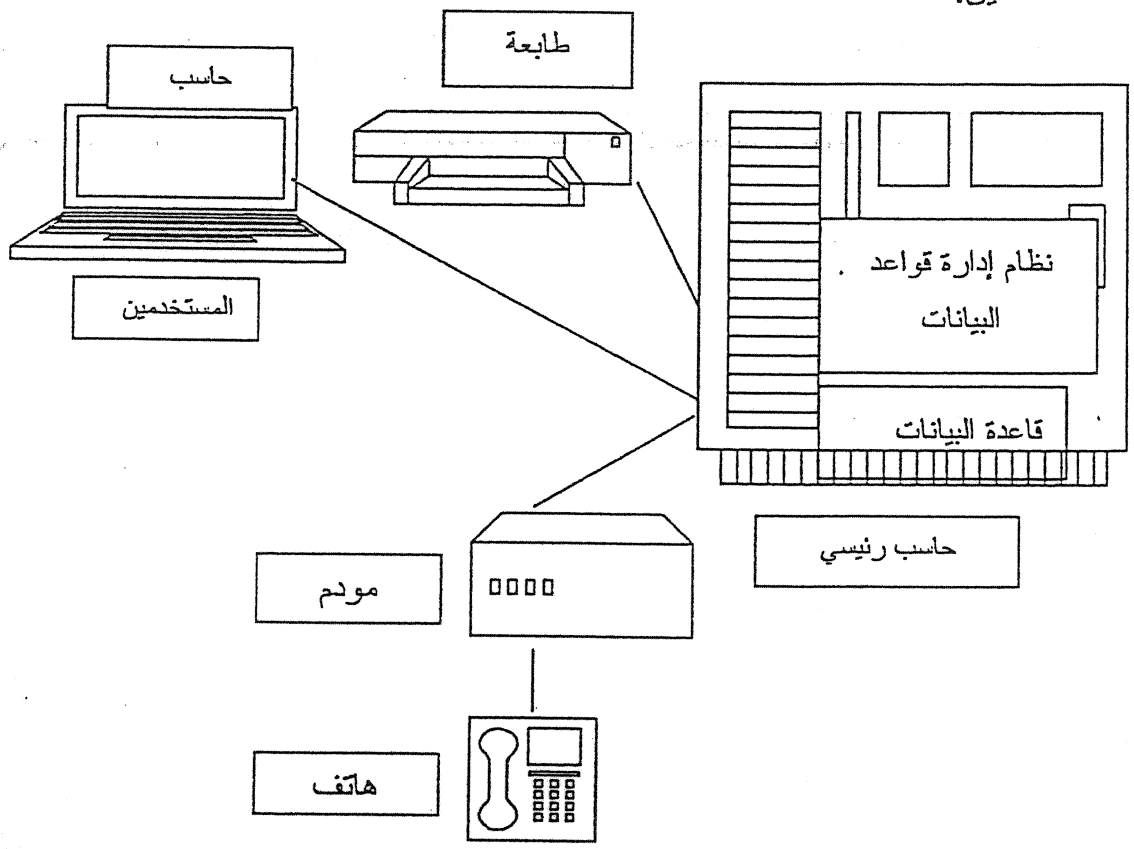
المعلومات هي أي شيء هام للمستخدم الشخص أو للمنظمة من اجل تسيير العمل الشخصي أو أعمال المنظمة.

ثالثاً- مكونات نظام المعلومات المحوسب:

يتكون نظام المعلومات المحوسب، باستخدام نظم إدارة قواعد البيانات، كما هو واضح في الشكل (١-٢) من المكونات التالية:

- قاعدة البيانات،

- البرمجيات،
- التجهيزات المادية،
- المستخدمين.



الشكل (١-٢) مكونات نظام المعلومات المحوسب

١ – قاعدة البيانات:

يمكن تعريف قاعدة البيانات بأنها مجموعة من البيانات أو المعلومات المتصلة، ذات العلاقات المتبادلة فيما بينها والمخزنة بطريقة نموذجية ودون تكرار، إنها التجميع المستمر للبيانات التي يتم استخدامها من قبل التطبيقات المختلفة في منظمة محددة.

إن نظام المعلومات يتكون من مجموعة مدخلات حول العمليات يتم تخزينها داخل النظام لتعالج وفق أساليب محددة من أجل الوصول إلى المعلومات المطلوبة مع تأمين الرقابة الكافية على أصول المنظمة.

تتكون قاعدة البيانات من بيانات حول موارد المنظمة (عاملين، مواد أولية، منتجات، عملاء، موردين..الخ) والعمليات(الأحداث، بيع، شراء، استلام، تسليم، نقل..الخ) التي تؤثر على هذه الموارد، حيث يمكن النظر للنظام على أنه مجموعة بيانات حول هذه الموارد والأحداث.

تقوم فكرة قاعدة البيانات على تخزين الصفات الهامة المتعلقة بالأحداث والموارد على شكل جدول، فيفرض أننا نريد تجميع البيانات حول المنتجات، العملاء، الفواتير..الخ، أمكننا عرض هذه البيانات في جداول.

في ظل قواعد البيانات الترابطية، يطلق على هذه الجداول اسم (Relation) رابطة، فقاعدة البيانات السابقة تتضمن رابطة العملاء ورابطة المنتجات ورابطة الفواتير ورابطة المبيعات. وتتضمن كل رابطة مجموعة الأسطر يطلق على الواحد منها سجل (Record)، ويتكون السجل بدوره من مجموعة حقول (Fields)، فالحقول المكونة لسجل المنتجات هي: (رقم المنتج، التوصيف، مستوى إعادة الطلب، الكمية في المخزن، تكلفة الوحدة).

إن ما يميز تنظيم قواعد البيانات هو أنها تؤمن التكامل بين البيانات المخزنة والتشارك في استخدام البيانات المخزنة.

يقصد بالتكامل (Integrated) توزيع بيانات النظام على عدة ملفات والمحافظة على الترابط بين البيانات مع أقل تكرار وحشو في البيانات، فرغم توزيع بيانات المبيعات على عدة جداول فإنه يمكن ربط هذه الجداول والوصول إلى المعلومة المطلوبة، مثلا يمكن الوصول إلى عدد الوحدات التي اشتراها أحد العملاء من المنتج "غسالة".

أما التشارك (Shared) فتعني أن عنصر البيانات الواحد يمكن أن يستخدم من قبل عدة مستخدمين بمعنى أن كل واحد من المستخدمين يملك القدرة على الوصول إلى البيانات المخزنة في نفس الوقت. ويقتضي هذا التشارك أن مستخدماً معيناً يكون مربوطاً عادة مع جزء صغير من قاعدة البيانات هو ذلك الجزء اللازم له لإنجاز عمله، فمثلاً يمكن لموظف المبيعات وموظف المخزن ومحاسب المدينين التعامل مع قاعدة البيانات السابقة، مع اختلاف الحقول التي

يحتاجونها فحقل مثل مستوى إعادة الطلب يهم موظف المخزن ولا يهم محاسب المدينين وحقل السعر يهم موظف المبيعات..الخ.

٢- نظام إدارة قاعدة البيانات:

بعد أن تعرفنا بشكل مبسط إلى قاعدة البيانات، سنتطرق الآن إلى نظام إدارة قواعد البيانات(*)(Database Management system)، أي ببساطة النظم التي تمكننا من بناء قاعدة البيانات السابقة واستخدامها.

نظم إدارة قواعد البيانات هي مجموعة من البرامج المصممة بشكل يمكن مصمم النظم من تنظيم ملفات البيانات وإدخال البيانات إلى الملفات، بالإضافة إلى احتوائها على مجموعة البرامج التي تمكن المستخدم من تصميم التقارير والنماذج والاستعلامات وكتابة التطبيقات، بحيث يتمكن ببساطة من الحصول على المعلومات المطلوبة من البيانات المخزنة في قاعدة البيانات، من دون أن يحتاج المستخدم إلى معرفة التفاصيل الدقيقة حول آلية تخزين البيانات على وسائط التخزين، ويتم تحقيق ذلك عبر طبقات من البرامج (Interface) التي تتولى مهمة الفصل بين المستخدم وبين شكل تخزين البيانات على وسائط التخزين.

يتضمن نظام إدارة قواعد البيانات مجموعة من البرامج، التي تقوم بأداء عدد من الوظائف، التي تسمح بإنشاء قاعدة البيانات وإدخال التعديلات عليها وإدخال البيانات إلى قاعدة البيانات وتحديثها، بالإضافة إلى توفير الإمكانية للمستخدم لإعداد التطبيقات التي تتناسب وعمله وحاجته على المعلومات، وأهم وظائف نظم إدارة قواعد البيانات الوظائف التالية:

- تعريف الروابط،
- معالجة البيانات،
- الاستعلام من قاعدة البيانات،
- ضمان حماية البيانات ونزاهتها،
- ضمان استعادة البيانات وعدم ازدواجية الاستخدام،

- قاموس البيانات.

وسوف نستعرض بشكل موجز كل هذه الوظائف لنظام إدارة قواعد البيانات:

أ- لغة تعريف البيانات(Data Description Language):

يحتوي نظام إدارة قواعد البيانات على مجموعة برامج يطلق عليها المترجم (Compiler)، تهدف إلى تمكين المستخدم من تعريف الروابط الخاصة بقاعدة البيانات، أي تحديد السجلات التي تنتمي إلى رابطة معينة، والحقول المكونة للسجل.

تستخدم لهذا الغرض لغة تعريف البيانات(Data Description Language) والتي تعرف اختصاراً بـ (DDL) التي تستخدم لتعريف وتوصيف الملفات والسجلات المكونة للملفات ضمن قاعدة البيانات تحت وجهات نظر المستخدم.

وتتضمن لغة تعريف البيانات الوظائف التالية:

- توصيف ملفات النظام والسجلات والحقول المكونة للملفات
- توصيف العلاقات بين الملفات
- توصيف شروط وقواعد التكامل
- توصيف أشكال التقارير التي تصدر عن النظام.

ب- لغة معالجة البيانات (Data Manipulation Language):

يجب أن يكون نظام إدارة قواعد البيانات قادراً على تنفيذ طلبات المستخدمين باسترجاع المعلومات من قاعدة البيانات وتحديثها وإضافة سجلات جديدة إلى قاعدة البيانات، وباختصار يجب أن يكون النظام قادراً على معالجة البيانات المخزنة قاعدة البيانات. يضاف مترجم آخر لنظام إدارة قاعدة البيانات يتولى تلقي طلبات المعالجة وترجمتها وتنفيذها هو مترجم لغة معالجة البيانات (Data Manipulation Language) والتي تسمى اختصاراً (*DML*) وهي عبارة عن لغة تنظيمية من أجل استخدامها في معالجة البيانات المخزنة ضمن قاعدة البيانات.

ومن الأمثلة على الوظائف التي يمكن أن تقوم بها لغة معالجة البيانات (*DML*):

- إضافة سجلات جديدة إلى ملف موجود
- فتح ملف معين موجود ضمن قاعدة البيانات
- إغلاق ملف مفتوح مسبقاً،
- قراءة بعض أو كل السجلات الموجودة ضمن ملف معين،
- تغيير محتويات الحقول الموجودة في سجل معين،
- دمج الملفات وإجراء الترابط بين الملفات،
- مسح سجل أو مجموعة سجلات من أحد الملفات،
- صياغة البرامج التي تسمح بإجراء عمليات المعالجة على البيانات مثل العمليات الحسابية والمنطقية وعمليات فهرسة وفرز السجلات في الروابط.

لغة الاستعلام (SQL): وتستخدم لطلب معلومات من قاعدة البيانات حيث كانت لغة معالجة البيانات قد استخدمت لتغيير محتويات قاعدة البيانات فتقوم لغة الاستعلام باسترجاع وتصنيف وترتيب وتحضير مجموعات فرعية من قاعدة البيانات للإجابة على استعلام المستخدمين. ومعظم أنظمة لغات الاستعلام تحتوي على حيادية وعدالة كبيرتان بالإضافة إلى سهولة الاستخدام في نفس الوقت.وتشغل الأوامر يُمكن المستخدم من الحصول على المعلومات التي يريدها من دون الرجوع إلى المبرمج.

تتضمن العديد من أنظمة إدارة قواعد البيانات **نظام لإعداد التقارير** وهو عبارة عن لغة مبسطة لإنشاء تقرير وبشكل نموذجي فإن المستخدمين الذين يحتاجون بعض عناصر البيانات يمكنهم كتابة التقرير وتحديد كيفية تنسيقه وسيقوم كاتب التقارير بعدها بالبحث في البيانات وفك ضغط عناصر بيانات محددة وطباعتها كمخرجات حسب التنسيق الخاص الذي طلبه المستخدم.

في العادة كافة المستخدمين يستطيعون الوصول إلى كلّ من لغة استعلام البيانات وكاتب التقارير لكن يجب تقييد الوصول إلى لغتا التوصيف والمعالجة للبيانات لهؤلاء الموظفين وحصرها فقط لمدير قاعدة البيانات والمبرمجين مما يساعدنا على تحديد عدد الأشخاص الذين يستطيعون تغيير قاعدة البيانات.

ج- ضمان حماية البيانات وسلامتها:

يقدم نظام إدارة بنك المعلومات من خلال لغة معالجة البيانات الإمكانية لتعريف وتصميم قواعد لضمان أمن البيانات(Data security) المخزنة في قاعدة البيانات، ويقصد بأمن البيانات حماية البيانات من الوصول غير المشروع إليها ويتم ذلك من خلال تحديد سلطات مستخدمي بنك المعلومات في الوصول إلى البيانات وتحديد نوع العمليات التي يسمح لهم إجراؤها على البيانات، فمثلاً لا يسمح لموظف المخزن بتعديل البيانات المتعلقة بأسطر الفاتورة في قاعدة بيانات المبيعات السابقة.

يقدم نظام إدارة بنك المعلومات الإمكانية لضمان سلامة البيانات (Data Integrity)، وتعني سلامة البيانات أن البيانات المخزنة يجب أن تكون صحيحة وخالية من التناقض في كل وقت من الأوقات وذلك من خلال:

- الرقابة على المدخلات من أجل منع تسجيل بيانات متناقضة أو غير صحيحة.
- ضمان أمن الملفات من الضياع أو التلف أو التزوير والتلاعب.
- حماية البيانات المخزنة من الوصول غير المشروع إليها عن طريق تعريف حقوق الوصول إلى البيانات.

فعلى سبيل المثال عندما يرغب أحد المستخدمين إدخال فاتورة جديدة إلى قاعدة بيانات المبيعات في جدول الفواتير، أن يدخل رقم العميل، بالرغم من عدم وجود رقم العميل المدخل في جدول العملاء، ففي مثل هذه الحالة تكون البيانات غير نزيهة، إذ كيف يمكن وجود لرقم العميل في جدول الفواتير من دون أن تكون البيانات المتعلقة بهذا العميل في جدول العملاء.

ج- **ضمان استعادة البيانات وعدم التداخل في التحديث:**

بما أن المنظمة تضع معظم بياناتها في قاعدة البيانات، لذلك يقدم نظام إدارة قاعدة البيانات الوسائل الضرورية من أجل إعادة إنتاج البيانات(Data Recovery) التي يصيبها الضرر في حال حدوث خطأ من قبل المستخدمين أو عطل في المكونات المادية للنظام بأقل عدد

ممكن من المعالجات، كذلك يقدم الإمكانية لتوليد نسخ احتياطية من الملفات لاستخدامها في حالات الطوارئ.

تنشأ مشكلة التداخل(Concurrency) كون نظم إدارة قواعد البيانات تسمح لعدة مستخدمين(برامج) بالوصل واستخدم عنصر البيانات في نفس الوقت، وفي مثل هذه الحالة تنشأ الحاجة إلى وجود آلية للرقابة على هذا التداخل، بحيث لا تحول عملية معينة تتم على البيانات من قبل المستخدم آ، من دون إتمام العملية المطلوبة من قبل المستخدم ب.

د- قاموس البيانات:

يحتوي نظام إدارة قواعد البيانات على وظيفة قاموس البيانات (Data Dictionary)، الذي يمكن أن يفهم على أنه قاعدة بيانات خاصة بالنظام وليس بالمستخدم، تتضمن بيانات حول البيانات المخزنة في قاعدة البيانات، لذلك يطلق عليه البيانات التحتية (Metadata)، إن هذه الوظيفة تمكن من تخزين مواصفات كل حقل من حقول قاعدة البيانات الأساسية مثل اسم الحقل، نموذج البيانات، حجم الحقل، فهرسة الحقل، شروط النزاهة الخاصة بالحقل، والتطبيقات التي يحق لها أن تتعامل مع هذا الحقل وحدود هذا التعامل، والربط بين الحقل في هذه الرابطة والروابط الأخرى وشروط هذا الربط..الخ.

ويتم تسجيل المعطيات التالية لكل مفردة من مفردات البيانات:

اسم المفردة: وهو الاسم الذي نستخدمه داخل النظام للتعبير عن عناصر البيانات مثل أن نسمي(رقم العميل) ب (Custnr) وأسم الزبون ب(Name).

تعريف المفردة: ويشمل تحديد معنى المفردة ومفهومها بالإضافة إلى تحديد نوع المفردة وطولها(أي عدد الرموز التي تشكل محتوى المفردة) مثل أن نحدد أن رقم الهاتف معرف على مجموعة الأعداد الصحيحة.

تحديد مصدر المفردة : طريقة الحصول على المفردة هل هي المدخلات أم المعالجة. فمصدر بيانات العملاء هي المدخلات أما رصيد العميل فيتم الحصول عليه من عمليات

المعالجة، بالإضافة إلى ذلك يتم تحديد قواعد التحقق على سبيل المثال إن رقم العميل يجب أن يكون محصوراً بين ١ و ١٠٠٠.

أين تستخدم : والمقصود بذلك تحديد القسم أو الدائرة التي تستخدم مفردة البيانات المذكورة والتقارير التي تستخدم فيها هذه المفردة . فمثلا يستخدم حقل الرصيد من قبل قسم الحسابات المدينة عند إعداد كشوف العملاء الشهرية.

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
رقم الزبون	- وحيد -يعرف الزبون	-سجل الزبائن -سجل تحليل المبيعات	قائمة أرقام الزبائن	١٠	رقمي	-تعديل ملف الزبائن -تعديل ملف المبيعات	-تقرير المبيعات -تقرير الزبائن -تقرير المديونية	لايوجد تقييد	لا يوجد
اسم الزبون	الاسم الكامل للزبون	سجل الزبائن	طلب الزبون الأولي	٢٠	حرفي	-تحديث ملف الزبائن -قائمة العمليات	-تقرير حالة الزبائن -التقرير الشهري	لا يوجد تقييد	لا يوجد

الجدول (١-١) قاموس البيانات

التحديث والصيانة: يجب تحديد كيفية تحديث مفردة البيانات وتوقيت عملية التحديث ودوريتها فمثلا تتم عملية تحديث رصيد العميل بعد كل عملية بيع أو سداد.

التخزين: وتتضمن تحديد أسماء الملفات التي تحتوي على مفردة البيانات وتحديد الوسط التخزيني الذي يخزن الملف عليه .

٣- **المكونات المادية:**

يقصد بالمكونات المادية(Hard Ware) التجهيزات الضرورية لبناء واستخدام بنك المعلومات وتتلخص المكونا ت الضرورية لبناء وتشغيل بنوك المعلومات، انظر الشكل(١-٢)، بالمكونات التالية:

- الحواسب بمختلف أنواعها من الحواسب الكبيرة (Main Frame) إلى الحواسب الشخصية (Personal Computer)
- وسائط التخزين المباشرة المناسبة مثل الأقراص المغناطيسية والأقراص الصلبة والأقراص المرنة..الخ.
- الأجهزة الطرفية المربوطة بالحاسب (*Terminal*) مثل الشاشات الآلات الطابعة من أجل إدخال البيانات إلى البنك واستدعاء المعلومات من البنك.
- قنوات الاتصال بين حواسب الشبكة وهي الوسائط التي تستخدم في نقل البيانات بين المكونات المادية للنظام الحاسوب مثل المودم ، الأسلاك ، المازج *Multiplexer* .

في بنوك المعلومات التي تكون البيانات فيها كبيرة الحجم وتعالج بكثافة مثل المحاسبة يجب أن تتصف المكونات المادية بالصفات التالية:

أ- طاقة تخزينية كبيرة للوحدة الواحدة بهدف تخفيض تكلفة التخزين.

ب- زيادة الاستقلالية بين تنظيم البيانات والمكونات المادية بهدف الوصول إلى درجة ثقة عالية .

٤- **المستخدمين:**

يتم التميز بين ثلاثة أنماط من مستخدمي قاعدة البيانات هم:

أ- **مدير قاعدة البيانات** (Database Administrator) :

حيث أنه من غير المسموح أن توجه بنوك المعلومات إلى إيفاء المتطلبات لمستخدم معين من بين المستخدمين، وإنما يجب أن يخدم جميع المستخدمين بنفس الدرجة، لذلك يقوم مدير

بتلك المعلومات بوظيفة تحقيق التوافق بين متطلبات المستخدمين والاستغلال الأمثل لطاقات وإمكانيات النظام.

تشمل مهام مدير قاعدة البيانات الوظائف التالية:

- تحديد متطلبات قواعد البيانات المطلوبة من برمجيات وتجهيزات،
- إدامة النظام والتنسيق بين متطلبات المستخدمين عند استخدام قواعد البيانات
- توفير الأمن والحماية لقواعد البيانات .
- إعادة تنظيم قاعدة البيانات عند الحاجة وتأمين إعادة البيانات إلى وضعها الطبيعي في حال حدوث الخطأ في العمليات مع البنك (Recovery).
- تحديد صلاحيات المستخدمين في الوصول وإجراء العمليات على قاعدة البيانات.
- الرقابة وضبط أداء النظام ضمن مقياس عمل مثالي.

ب- مبرمجي التطبيقات (Applications programmers):

يقوم مبرمجي التطبيقات بإعداد البرامج التي تقوم بمعالجة البيانات المخزنة في قاعدة البيانات من أجل أن تقدم الإمكانية للمستخدم النهائي لإضافة البيانات ،تعديلها واسترجاعها . ويقوم مبرمجو التطبيقات بكتابة هذه البرامج بإحدى لغات البرمجة التي عادة ما تكون مربوطة مع لغة معالجة البيانات (DML) ويتم عادة من خلال هذا البرنامج تصميم الشاشات والبرامج الحوارية بين المستخدم والحاسب. وبرنامج التطبيقات هو مجموعة من فعاليات المعالجة المترابطة والمغلقة.

تشمل مهام مبرمجي التطبيقات المهام التالية:

- تحويل وترجمة تصميم قاعدة البيانات إلى قاعدة بيانات فعلية باستخدام إحدى اللغات المناسبة
- تنفيذ الأنظمة والبرمجيات والتأكد من صحتها وخلوها من التناقض
- صياغة شاشات التخاطب والإدخال والإخراج التي تحتاجها نظم قواعد البيانات وتنفيذها.

- صياغة أنماط وأشكال النماذج والتقارير المطلوبة وتنفيذها.

ج- **المستخدمين النهائيين** (End Users):

هم المستخدمون الذين يتحاورون مع النظام من خلال المحطة الطرفية بواسطة إما برنامج التطبيقات الوارد في الفقرة السابقة أو بواسطة اللغات الموجودة في نظام إدارة بنك المعلومات مثل (SQL)، التي يقدم بعض الأوامر البسيطة مثل (SELECT, INSERT,UPDATE,DELETE ..etc.) ليتمكن المستخدم النهائي من التعامل مع قاعدة البيانات مباشرة.

ثالثاً- بناء (معمارية) نظام قاعدة البيانات Database System Architecture :

إحدى المزايا الهامة لنظم إدارة قواعد البيانات هي فصلها بين ثلاث مشاهد (Views) لقاعدة البيانات، هي المشهد الخارجي والمشهد المنطقي والمشهد الداخلي، وهذا الفصل هو الذي يمكن مستخدم قاعدة البيانات من تصميم التطبيق الخاص به من دون الحاجة إلى معرفة كيفية تخزين البيانات على وسائط التخزين المختلفة. وسوف نستعرض هذه المستويات التي يطلق عليها معمارية نظام قاعدة البيانات (Database System Architecture)، والتي تظهر في الشكل (١-٣).

١- المستوى الخارجي (External Level):

المستوى الخارجي هو مستوى المستخدم المفرد مثل المستخدم النهائي أو مبرمج التطبيقات. وبما أن كل مستخدم مهتم فقط بمقطع من قاعدة البيانات الكلية، وهذا المقطع يمثل وجهة نظر المستخدم على قاعدة البيانات، لذلك يقوم كل مستخدم من المستخدمين بتحديد المعطيات الضرورية ليصل إلى النتائج المرغوبة عبر القيام بمجموعة من العمليات الحسابية والمنطقية والإدخال والإخراج، وبالتالي فان إجراء هذه العمليات يتطلب وجود لغة تمكن من إجراء العمليات على ملفات قاعدة البيانات، هذه اللغة هي لغة معالجة البيانات (DML) بالنسبة لمبرمجي التطبيقات أوSQL .

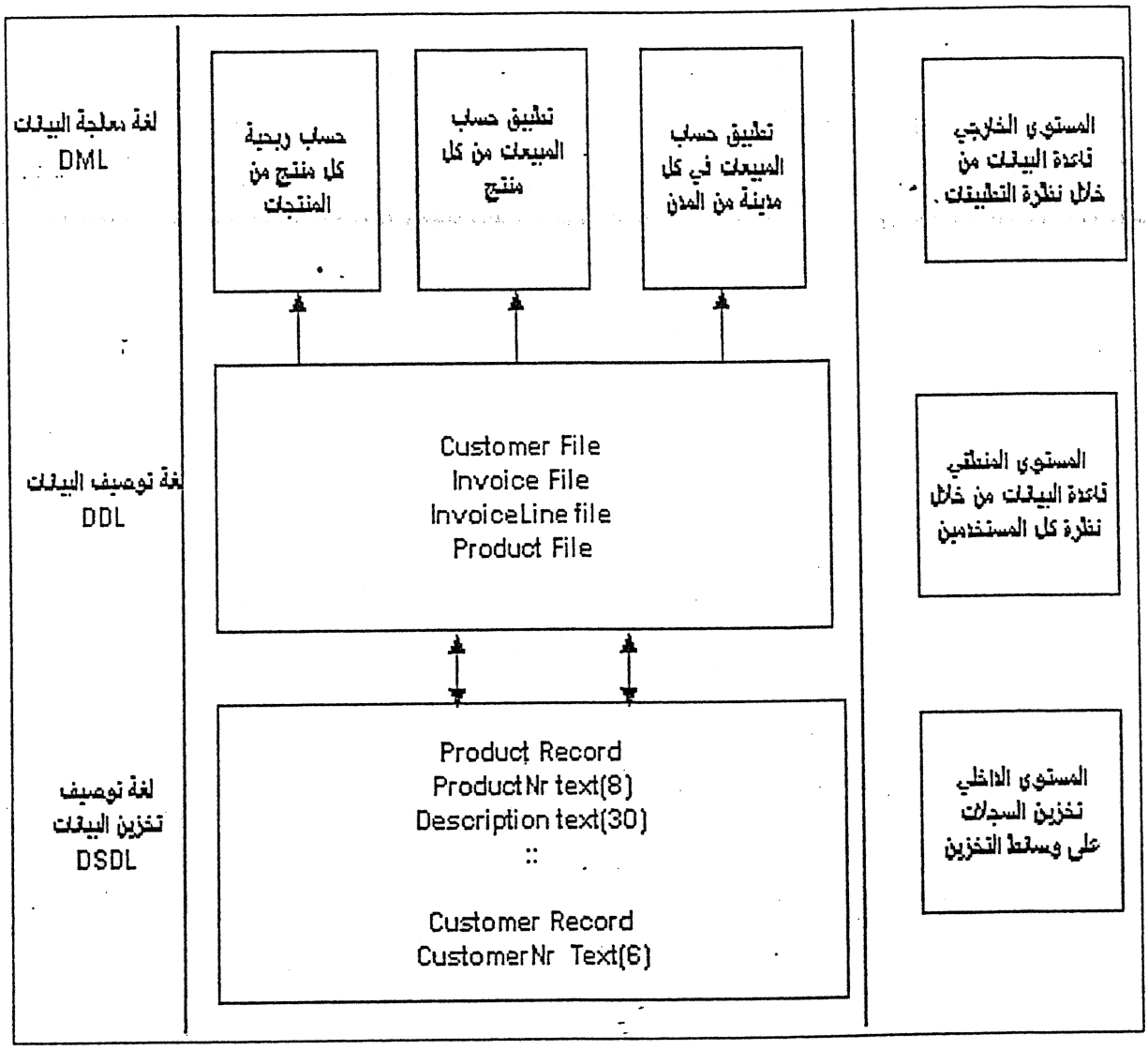
٢-المستوى المنطقي (Conceptual Level):

يتم في هذا المستوى توصيف الملفات من حيث البنية المنطقية. ويتم في هذه المرحلة تحديد الملفات الضرورية لبناء قاعدة البيانات وتحديد السجلات التي تتكون منها هذه الملفات من حيث حقول السجل والبيانات التي سوف تخزن في حقول السجل، وذلك باستخدام لغة توصيف البيانات (DDL)، ويتم وضع هذا التوصيف للملفات بغض النظر عن أشكال تخزينها على وسائط التخزين المغناطيسية وبغض النظر عن المستخدم الذي سوف يستخدم هذه الملفات.

إنها بناء الجداول التي تنسجم مع حاجات كافة مستخدمي النظام بشكل إجمالي وليس حسب حاجة كل مستخدم بشكل مستقل.

تعد هذه المرحلة من أهم المراحل عند بناء قاعدة البيانات إذ أنها تتضمن الفعاليات التالية :

- تحليل المشكلة وتحديد الحاجة إلى المعلومات وجريان البيانات والمعلومات ضمن المنظمة .
- تحديد البيانات التي يجب أن تخزن في قاعدة البيانات من اجل الوصول إلى المعلومات المطلوبة من أجل إشباع حاجة الإدارة إلى المعلومات .
- تحديد العلاقات بين الملفات التي سوف يتم تخزينها في قاعدة البيانات .
- تعريف التوافق المنطقي بين البيانات من اجل ضمان خلو البيانات المخزنة في قاعدة البيانات من التناقض .
- تعريف الأشخاص أو الوحدات ضمن المنظمة التي تملك حق الوصول والإطلاع على البيانات المخزنة ضمن قاعدة البيانات .



الشكل(١-٣) معمارية نظام قاعدة البيانات.

٣-المستوى الداخلي (Internal Level):

هو مستوى البناء المادي (الفيزيائي) للملفات حيث يوجد العديد من أساليب تخزين البيانات على وسائط التخزين. ويقصد بآلية تخزين البيانات على وسائط التخزين كيفية تخزين السجلات المكونة لأحد الملفات على وسيط التخزين من أجل الوصول إلى أفضل أداء لقاعدة البيانات واستخدام أفضل حيز تخزين واستخدام تراكيب البيانات المناسبة بالإضافة إلى توفير آليات التخاطب مع نظم التشغيل في تخزين البيانات والسجلات واسترجاعها من وإلى مواقع

تخزينها، حيث ويوجد هناك العديد من أساليب تخزين البيانات على وسائط التخزين مثل الملفات المفهرسة والملفات المباشرة.

ويمكن تخليص وظائف هذا المستوى بالوظائف التالية:

- تحديد أماكن التخزين والفهارس للبيانات
- وصف السجلات لغايات التخزين وتحديد حجمها.
- حفظ البيانات وتشفيرها
- تحديد التراكيب الداخلية للبيانات وهياكلها.

تقوم فلسفة بنوك المعلومات على تحرير المستخدم من هذه المهام عن طريق قيام نظام إدارة قواعد البيانات بأداء وظيفة تخزين البيانات الملفات على وسائط التخزين المختلفة.

من اجل تحقيق وتنفيذ نظام المعلومات بالشكل المذكور أعلاه يحتاج المرء إلى نظام بنك معلومات ذي إمكانيات كبيرة وكذلك إلى لغة معالجة بيانات يسهل استخدامها على غير المختصين في الحاسوب مثل العاملين في أقسام المحاسبة والأقسام المالية وذلك من اجل أن يتمكنوا من استخدام بنك المعلومات لحل المشاكل التي تواجههم ويصلوا إلى المعلومات التي يرغبون بالوصول إليها.

رابعاً: مزايا نظم إدارة قواعد البيانات:

إن استخدام الأساليب التقليدية لمعالجة المعلومات(نظم الملفات) في تشغيل نظم المعلومات الإدارية في المنظمات نتج عنه مجموعة من السلبيات التي يمكن تلخيصها كما يلي:

١- يقوم كل تطبيق بإدارة وتكوين ملفاته الخاصة ، وبما أن التطبيقات مرتبطة ببعضها فإن ذلك يؤدي إلى تخزين نفس البيانات في العديد من الملفات العائدة للتطبيقات المختلفة ، مما يؤدي إلى نشوء حشو وتكرار في تخزين نفس عناصر البيانات في عدة ملفات ، مما يعيق عمليات التحديث ويجعلها طويلة.

٢- إن الملفات تكون مصممة من قبل برنامج التطبيق وبما يتلاءم مع احتياجات التطبيق وبالتالي فان البرنامج هو الذي يحدد هيكل الملفات وتفاصيلها وبالتالي فان أي تعديل في بنية هذه الملفات سوف يتطلب تعديل البرنامج وكذلك فان التعديل في البرنامج قد يتطلب تعديل الملفات وتابعية الملفات للبرامج . .

٣- إن خطوات المعالجة الضرورية لحل مشكلة معينة " الخوارزمية ،" أو الوصول إلى معلومات معينة يتم تحديدها بعد ظهور المشكلة أو بنشوء الحاجة إلى المعلومة المطلوبة وبالتالي فان الوصول إلى تلك المعلومة يتطلب إعداد الخوارزمية وكتابة البرنامج الذي سوف يقوم بأداء هذه المهمة وذلك يتطلب وقتاً مما يعني التأخير في حل المشكلة المطلوبة.

٤- إن الرقابة على صحة ونوعية البيانات المعالجة والمدخلة هي من اختصاص برنامج التطبيقات نفسها ، والمبرمجون عادة هم الذين يحددون نوع ودرجة الرقابة المنطقية التي يؤديها البرنامج . إن ذلك قد يكون كاف في ذاته تحت وجهة نظر واعتبارات التطبيق ، أما بالنسبة لنظام المعلومات بشكل خاص في المنظمة فان ذلك يعتبر غير كافي . لذلك لا بد من وجود وسائل وخطوات رقابية إضافية لضمان صحة البيانات بحيث تكون البيانات المخزنة في عدة ملفات خالية من التناقض فيما بين هذه الملفات وتعبر عن الواقع الفعلي للمنظمة.

٥- إن وضع تطبيقات جديدة على الملفات الموجودة في النظام هو أمر مكلف على الأغلب إذ أن ذلك يتطلب إعادة هيكلة الملفات الموجودة أو إنشاء ملفات جديدة تتلاءم والتطبيق الجديد.

إن استخدام نظم إدارة قواعد البيانات في تطوير نظم المعلومات يحقق في النظام المطور المزايا التالية:

٦- من خلال النظر إلى قاعدة البيانات من ثلاثة مناظير تترفع استقلالية البرامج عن البيانات، بحيث أن التعديلات في تصميم الملفات(الجداول) لا تقود إلى تعديلات في البرامج(الاستعلامات مثلاً) وكذلك فان التعديلات في البرامج لا تقود بالضرورة إلى تعديلات الملفات القائمة.

١- إن كون البيانات غير تابعة ومرتبطة ببرامج التطبيقات مما يمكن من إعداد برامج تقويم لمخزون البيانات في وجهات نظر متعددة - وبالتالي وجود إمكانية لوضع حلول للمشاكل والوصول إلى معلومات تكون غير مخطط لها أو متوقعة عند تصميم وبناء قاعدة المعطيات.

٢- إن كون قاعدة البيانات تسمح بالتشاركية في استخدام البيانات المخزنة، فإنه يمكن تصميم وتخزن قاعدة البيانات على أساس احتياجات المنظمة بشكل كامل،مما يؤدي إلى التقليل في تكرار تخزين البيانات وبالتالي يقلل من حجم البيانات الواجب تخزينها وتوثيقها.

٣- إن تصميم قاعدة البيانات بشكل موحد حسب احتياجات المنظمة بشكل كامل يؤدي إلى وجود ضوابط رقابية واحدة لكل البيانات المخزنة في قاعدة المعطيات مما يعني عدم وجود تناقض بين الملفات المخزنة في قاعدة البيانات.

٤- يقدم نظام إدارة بنك المعلومات من خلال لغة معالجة البيانات الإمكانية لتعريف وتصميم قواعد لضمان أمن البيانات المخزنة في قاعدة البيانات. ويقصد بأمن البيانات حماية البيانات من الوصول غير المشروع إليها ويتم ذلك من خلال تحديد سلطات مستخدمي بنك المعلومات في الوصول إلى البيانات وتحديد نوع العمليات التي يسمح لهم إجراؤها على البيانات.

٥- يقدم نظام إدارة بنك المعلومات الإمكانية لضمان سلامة البيانات(Data Integrity) . وتعني سلامة البيانات أن البيانات المخزنة يجب أن تكون صحيحة وخالية من التناقض في كل وقت من الأوقات.

٦- استعادة البيانات وإنشاء نسخ احتياطية من الملفات الموجودة في قاعدة البيانات: بما أن المنظمة تضع معظم بياناتها في قاعدة البيانات . لذلك من يقدّم نظام إدارة بنك المعلومات الوسائل الضرورية من أجل إعادة إنتاج البيانات التي يصيبها الضرر في حال حدوث خطأ من قبل المستخدمين أو عطل في المكونات المادية للنظام اقل عدد ممكن من المعالجات . كذلك يقدم الإمكانية لتوليد نسخ احتياطية من الملفات لاستخدامها في حالات الطوارئ .

أسئلة وتمرينات الفصل الأول

السؤال الأول: أختر الإجابة الصحيحة في كل مما يلي:

١- تحديد متطلبات كافة المستخدمين في قاعدة البيانات هي مهمة:

أ- مدير قاعدة البيانات

ب- مبرمجو التطبيقات

ج- المستخدم النهائي

د- كل هؤلاء.

٢-إن تصميم البرنامج وتنفيذه هي مهمة:

أ- مدير قاعدة البيانات

ب- مبرمجو التطبيقات

ج- المستخدم النهائي

د- كل هؤلاء.

٣- بناء العلاقات بين الروابط تتم باستخدام:

أ- لغة توصيف البيانات DDL

ب- لغة معالجة البيانات DML

ج- لغة الاستعلام SQL

د- لغات البرمجة الأخرى.

٤- توصيف كيفية تخزين البيانات في قاعدة البيانات على وسائط التخزين هي جزء من:

أ- المستوى الداخلي

ب- المستوى المنطقي

ج- المستوى الخارجي

د- كل هذه المستويات.

٥-تحديد ذلك الجزء من قاعدة البيانات الذي يهم تطبيق محدد هو جزء من:

أ- المستوى الداخلي

ب- المستوى المنطقي

ج- المستوى الخارجي

د- كل هذه المستويات.

٦- توصيف السجلات والحقول المكونة للسجلات هي من مهام:

أ- المستوى الداخلي

ب- المستوى المنطقي

ج- المستوى الخارجي

د- كل هذه المستويات.

٧- التعديلات في تصميم البيانات لا يقود إلى تعديل البرامج هي خاصية:

أ- المرونة

ب- الاستقلالية

ج- التشاركية

د- التكامل

٨- توزيع البيانات على عدة ملفات هي خاصية:

ه- التشاركية

و- التكامل

ز- النزاهة

ح- السرية

٩- تحديد صلاحية الوصول إلى المعلومات وإجراء التعديلات عليها هي ضمن خاصية:

ط- الاستقلالية

ي- السرية

ك- النزاهة

ل- التكامل

١٠- ضمان صحة البيانات وخلوها من التناقض هي خاصية:

م- المرونة

ن- الاستقلالية

س- النزاهة

ع- التكامل.

السؤال الثاني: أجب عن الأسئلة التالية:

١- ما هي مزايا استخدام نظم إدارة قواعد البيانات؟

٢- ما هي مساوئ استخدام أنظمة الملفات في بناء نظم المعلومات؟

٣- ما هي وظائف مدير قاعدة البيانات؟

٤- ما الفرق بين قاعدة البيانات ونظام إدارة قواعد البيانات؟

٥- ما هي وظائف نظام إدارة قواعد البيانات؟

السؤال الثالث:

معظم نظم إدارة قواعد البيانات تتضمن ثلاثة أنواع من اللغات هي: لغة توصيف البيانات (DDL)، لغة معالجة البيانات (DML)، ولغة الاستعلام(SQL)، حدد اللغة التي تستخدم في كل من الحالات التالية:

أ- تعريف البنية المنطقية للبيانات.

ب- المراجع الداخلي يحتاج إلى تقرير حول الفواتير المباعة في الاسبوع الماضي.

ج- أحد المبرمجين يريد إعداد برنامج يتولى تحديث جدول الأصول الثابتة في قاعدة البيانات.

د- أحد المبرمجين يريد إضافة حقل جديد إلى جدول العاملين في قاعدة البيانات.

تصميم قواعد البيانات العلائقية

أهمية تصميم قواعد البيانات:

إن عملية بناء قاعدة بيانات جيدة لا يأتي بتلك السهولة، إذ لابد من بذل جهد كبير للحصول على قاعدة بيانات جيدة. والتصميم الجيد لقاعدة البيانات يسهل عملية استخدام وإدارة هذه القاعدة أما التصميم السيئ فسيؤدي إلى تكرار البيانات(ويعني وجود نفس البيانات في أكثر من مكان) وبالتالي تصعب عملية الحفاظ على توافقية البيانات وعادة ما يؤدي تكرار البيانات إلى نتائج غير صحيحة عند طلب تلك البيانات من تلك القاعدة وهذا بدوره يؤدي إلى أن أي قرارات إدارية وكذلك أي تخطيط مستقبلي سيكون خاطئا لاعتماده على معلومات غير صحيحة.

دورة الحياة لنظام قاعدة البيانات:

١ - **الدراسة المبدئية للنظام القائم وتشمل ما يلي:**

أ - تحليل الوضع الحالي للمؤسسة ومعرفة طبيعة الإجراءات المستخدمة والتعليمات وقواعد العمل .

ب - تحديد المشاكل التي تواجه النظام المستخدم وكذلك القيود المادية مثل الطاقة البشرية والتمويل المتوفر لتطوير أو استبدال النظام الحالي .

ج - تحديد الأهداف الواجب تحقيقها والمزايا المطلوبة في النظام الجديد.

٢ - **تصميم قاعدة البيانات**: وتعتبر هذه المرحلة من أهم المراحل في دورة حياة النظام إذ لابد من بذل جهد كبير لتصميم النظام للوصول إلى نظام جيد وتؤدي الأهداف المرجوة من عمل النظام وتشمل عملية التصميم ما يلي:

أ - بناء نموذج المفاهيم وتشمل هذه العملية عدة خطوات(سنتطرق إلى هذه العملية بالتفصيل في الفصول اللاحقة):

١. تحليل البيانات ومتطلبات المستخدمين والإجراءات المطلوبة
٢. تعريف وتحديد الكيانات وخصائصها وعلاقتها مع بعضها وكذلك وضعها في الصيغة المعيارية.
٣. رسم مخطط المفاهيم وهو عبارة عن نموذج رسومي يوصف كيانات النظام وعلاقتها مع بعضها.
٤. تعديل النموذج بحيث يشمل الإجراءات الرئيسة، وقواعد عمليات الإضافة والتعديل والحذف على البيانات والتقارير، والشاشات، ومقدار التشاركية و توافقية البيانات....

ب - اختيار نظام إدارة قاعدة البيانات (DBMS).

ج - تحويل نموذج المفاهيم إلى نموذج داخلي بالاعتماد على نظام إدارة قاعدة البيانات(DBMS).

د - التصميم المادي وتتم خلاله عملية وضع مواصفات التخزين والوسائط المستخدمة في عملية التخزين وطرق الوصول للبيانات بالاعتماد على نظام إدارة قاعدة البيانات (DBMS).

٣ - **تنفيذ النظام:** وخلال هذه المرحلة تتم عملية إنشاء الجداول وكتابة جميع البرامج اللازمة لتنفيذ متطلبات النظام من الشاشات المختلفة و التقارير المطلوبة ... .

٤ - **عملية الفحص والتقييم للنظام وتشمل :**

أ - فحص قاعدة البيانات والتأكد من عملها بشكل صحيح.

ب -تقييم عمل البرامج والتطبيقات المستخدمة.

٥ - **تطبيق النظام في مكان العمل:** وتشمل هذه العملية عمليات إنشاء الجداول والمستخدمين والصلاحيات... وتحميل جميع البرامج والتطبيقات وتنفيذها في البيئة الحقيقة التي يجب أن يعمل بها النظام.

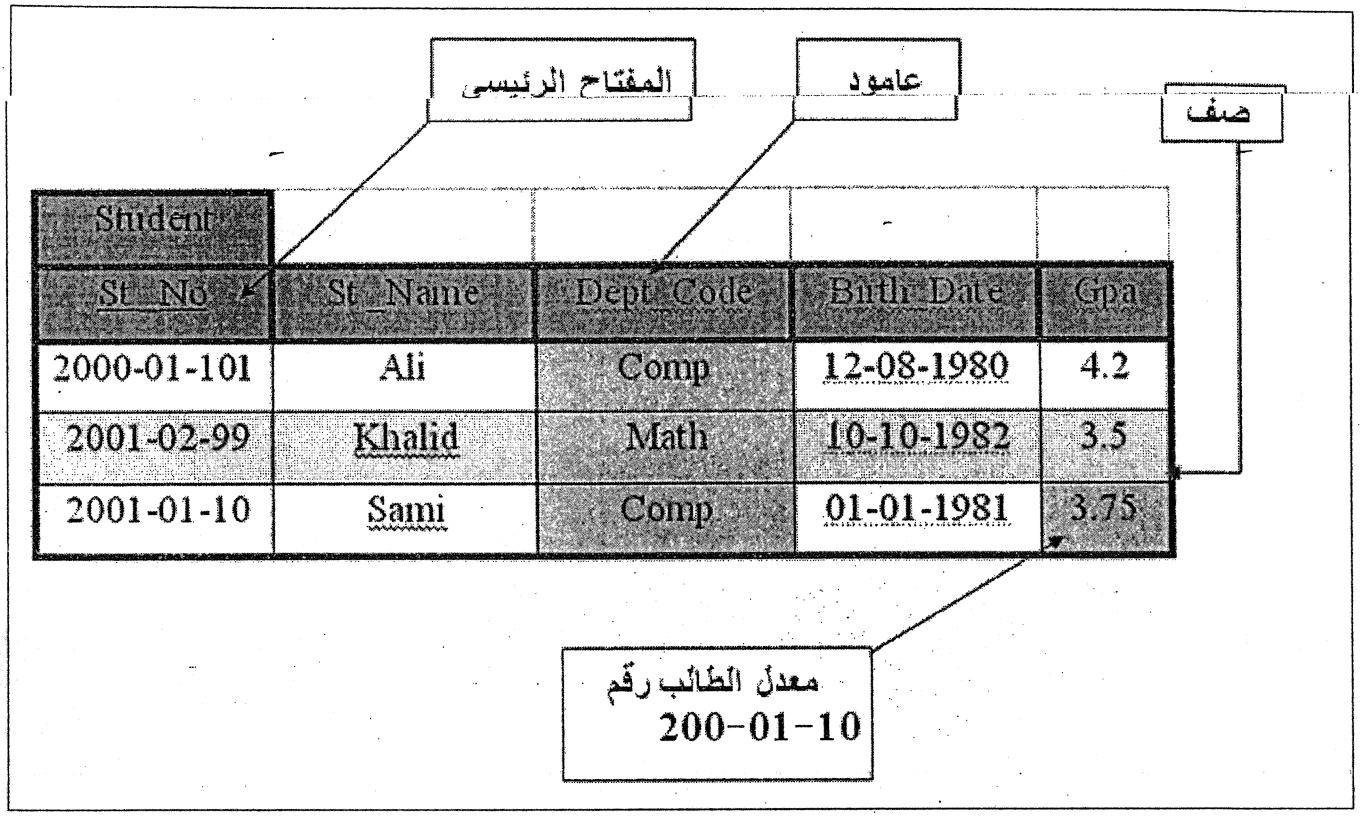
٦ -**متابعة عمل النظام:** وهذه العملية تستمر طيلة فترة حياة النظام للتأكد من عمله بشكل صحيح وكذلك تعديل النظام ليتواءم مع المتطلبات الجديدة لبيئة العمل مثل تغير القوانين والأنظمة وقواعد العمل.

قاعدة البيانات العلائقية :

بدأ نشوء مفهوم قواعد البيانات العلائقية عام ١٩٧٠ عندما قدم العالم Codd اقتراحا لهذا النموذج والذي تم بناؤه على نظريات الجبر العلائقي ومن هنا برزت قوة هذا النموذج وسرعة انتشاره فيما بعد. ففي مطلع الثمانينات بدأت الكثير من الشركات بتبني هذا النموذج وتطبقيه، فنلاحظ الآن أن معظم أنظمة قواعد البيانات الموجودة في الأسواق تتوافق مع هذا النموذج. وتتلخص فكرة النموذج في النظر إلى قاعدة البيانات على أنها مجموعة من الجداول (Tables) أو علاقات تسمى (Relations) ومن هنا جاءت تسمية النموذج وكل جدول يجب أن يكون له اسم (لا يوجد أكثر من جدول يحمل نفس الاسم).والعلاقة هي عبارة عن مصطلح رياضي وتمثل جدولاً ذا بعدين (صفوف وأعمدة)، ولا توجد هناك أهمية لترتيب الصفوف أو الأعمدة.حيث تمثل الصفوف مجموعة سجلات الجدول(Records or Tuple) وتمثل الأعمدة الصفات لهذا الجدول (Attributes) ويجب أن يكون لكل صفة مجال (Domain) من القيم التي يمكن أن يحتويها هذا العامود. وترتبط هذه الجداول مع بعضها بواسطة روابط..ويجب أن يكون لكل جدول مفتاح رئيس (Primary Key) لتمييز الصفوف عن بعضها والنقطة التي تمثل تقاطع الصف مع العمود (الصفة) تمثل قيمة لهذا الصف.و سنقوم في بقية أجزاء هذه الوحدة بتقديم وصفاً لقواعد البيانات العلائقية (Relational Database) من حيث مكوناتها وأهم خصائصها.

الجدول التالي يمثل معلومات الطالب (Student) في قاعدة بيانات إحدى الجامعات

- اسم الجدول Student
- كل صف يمثل معلومات تخص طالباً واحد فقط.
- المفتاح الرئيس للجدول هو St _No كل طالب يجب أن يكون له رقم مختلف عن بقية الطلاب.
- الصفة Dept_Code تمثل القسم الذي ينتمي إليه أي طالب .
- نقطة تقاطع الصفة (Gpa) العمود مع الصف الثالث تمثل المعدل التراكمي للطالب رقم ٢٠٠١ - ٠١ -١٠ .
- مجال القيم:كل صفة يجب أن يكون لها مجال ثابت من القيم فمثلا Gpa يجب أن تحتوي على رقم حقيقي بين ١.. ٥. القسم Dept_Code يجب أن يكون أحد الأقسام الدراسية الموجودة في الجامعة.



Student

St_No	St_Name	Dept_Code	Birth_Date	Gpa
2000-01-101	Ali	Comp	12-08-1980	4.2
2001-02-99	Khalid	Math	10-10-1982	3.5
2001-01-10	Sami	Comp	01-01-1981	3.75

- لا توجد هناك أهمية لترتيب الصفوف أو الأعمدة.فمثلا يمكن أن يكون الجدول السابق على الشكل التالي:

Student

St_No	St_Name	Gpa	Birth_Date	Dept_Code
٢٠٠١-٠١-١٠	Sami	٣,٧٥	١٩٨١-٠١-٠١	Comp
٢٠٠١-٠٢-٩٩	Khalid	٣,٥	١٩٨٢-١٠-١٠	Math
٢٠٠٠-٠١-١٠١	Ali	٤,٢	١٩٨٠-٠٨-١٢	Comp

مفاتيح الجداول (العلاقات):

تعتبر المفاتيح من أهم خصائص قواعد البيانات العلائقية حيث إنها تكون المميز لجدول معين من جهة والرابط الذي يربط الجداول المختلفة مع بعضها من جهة أخرى . ويمكن تقسيم المفاتيح في قواعد البيانات العلائقية إلى عدة أقسام :

أ - المفتاح الأعظم (Super Key): وهو صفة أو مجموعة من الصفات التي يمكن أن تميز الصف في الجدول عن بقية الصفوف الأخرى .فمثلا هذه المجموعة من الصفات يمكن أن تكون مفتاحا أعظم.

St_No
St_No, St _Name
St_No ,dept_code

ب - المفتاح المرشح (Candidate Key) : وهو الصفة (أو المجموعة من الصفات) التي يمكن اختيارها كمفتاح رئيس للجدول ويجب أن لا يكون هناك أكثر من صف له نفس القيمة لهذه الصفة أو الصفات وكذلك يجب أن يكون له قيمة (ليس Null) .

ولكن كما لاحظنا فإن St_No, St _Name هي مفتاح أعظم ولكنه ليس مفتاحا مرشحا ليكون مفتاحا رئيسا لأن St_No وحدة يكفي لتمييز أي صف عن بقية الصفوف، .لذلك فإن St_No يعتبر مفتاحا مرشحا ليكون مفتاحا رئيسيا .

ج - المفتاح الرئيس (Primary Key) : وهو المفتاح الذي تم اختاره من مجموعة المفاتيح المرشحة ليكون محددا لكل صف في الجدول .يمكن أن نختار St_No ليكون مفتاحاً رئيسا .

د - المفتاح الثانوي : هو عبارة عن صفة أوصفات تستخدم لغايات الاسترجاع ، فمثلا لو كان لدينا جدول يحتوي على قائمة بالعملاء فالمفتاح الرئيس هو رقم العميل Customer _id

ولكن إذا أردنا أن نسترجع رقم هاتف عميل معين (ولكن من سيحفظ أرقام العملاء ؟!) ففي هذه الحالة عادة ما يستخدم الاسم في عملية البحث وليس الرقم ،فيتم اختيار اسم العميل كمفتاح ثانوي .

Customer id	Customer name	tel	Address

هـ - المفتاح الأجنبي (Foreign Key) : وهو صفة أو صفات تشير إلى مفتاح رئيس أو قيمة غير مكررة(Unique) في جدول آخر فمثلا تمل الصفة (Dept_Code) في جدول المتدرب (Student) مفتاح أجنبيا(Foreign Key) لجدول الأقسام (Department)

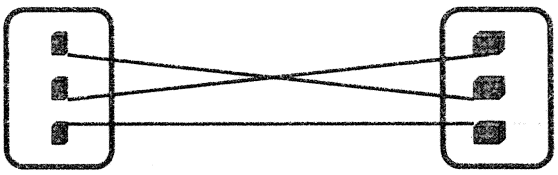
Student				
St _No	St _Name	Gpa	Birth_Date	Dept_Code
2001-01-10	Sami	3.75	01-01-1981	Comp
2001-02-99	Khalid	3.5	10-10-1982	Math
2000-01-101	Ali	4.2	12-08-1980	Comp

Department	
Dept_Code	Dept_name
Comp	Computer
Math	Mathematics

الارتباط بين الجداول (العلاقات):

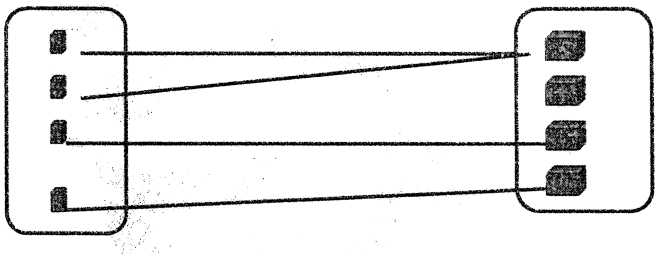
وتمثل الدرجة التي ترتبط بها الجداول مع بعضها فيجب أن تحدد هذه الروابط بشكل واضح لمعرفة كيفية ارتباط هذه الجداول مع بعضها . هناك ثلاث درجات لارتباط الجداول :

١. **واحد ــ واحد (١:١)**: وهذا يعني أن قيمة واحدة في الجدول الأول تقابل قيمة واحدة فقط في الجدول الثاني



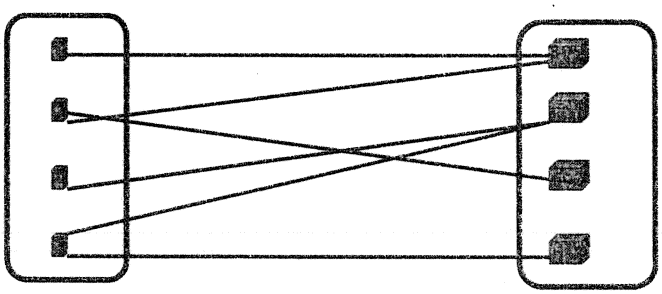
فمثلا يمكن أن نحدد على سبيل المثال أن لكل شخص جواز سفر واحد فقط وأن جواز السفر يعود لشخص واحد فقط .

٢. **واحد ـ متعدد أو متعدد ـ واحد** (١ : N أو N : ١) وهذا يعني أن قيمة في الجدول الأول تقابل قيمة في الجدول الثاني وأن القيمة في الجدول الثاني يمكن أن يقابلها قيمة أو أكثر في الجدول الأول.



فمثلا يجب أن يتبع المتدرب لقسم واحد فقط وفي الوقت نفسه يمكن أن يكون هنالك أكثر من طالب ينتمي لهذا القسم .

٣. **متعدد ـ متعدد(N:N)** : وهذا يعني أن قيمة في الجدول الأول تقابل قيمة أو أكثر في الجدول الثاني وأن القيمة في الجدول الثاني يمكن أن يقابلها قيمة أو أكثر في الجدول الأول.



فمثلا يمكن للطالب أن يسجل في أكثر من شعبة وكذلك الشعبة يمكن أن يسجل فيها أكثر من طالب.

نموذج الكيانات و العلاقات

مقدمة :

إن هدف عملية التصميم هو الوصول إلى فهم صحيح للنظام للمساعدة في عملية تطوير هذا النظام، وهذا ليس بالأمر السهل إذ لابد من وجود مقياس صحيح للحكم على هذا الفهم. ومن هنا برزت الأهمية لاستخدام العديد من الأدوات التي تساعد المصمم لوضع التصور والفهم الصحيحين لعمل هذا النظام.ومن هذه الأدوات استخدام النماذج التمثيلية التي تصف مكونات النظام وكيفية ارتباطها مع بعضها.وسنقوم في هذا الفصل بدراسة كيفية تمثيل البيانات باستخدام **نموذج الكيانات والعلاقات** Entity **Relationship (ER)Diagram**.

النماذج :

ما هو النموذج ؟

النموذج عبارة عن وصف رسومي (تمثيلي) لوصف الحقائق التي لا يمكن رؤيتها مباشرة.

وبعبارة أخرى هو وصف مجرد للكائنات الحقيقة. **نموذج البيانات** هو عبارة عن تمثيل بسيط لوصف تراكيب البيانات المعقدة في واقع الحياة العملية على شكل رسومي دون النظر إلى مكان وكيفية تخزين أو الوصول إلى هذه البيانات.ويستخدم هذا النموذج كوسيلة اتصال مابين المصمم من جهة وبين المبرمجين والمستخدمين من جهة أخرى. إذ حتى لو كان لدينا العديد من المبرمجين المحترفين فلا نستطيع الحصول على نظام جيد دون أن يكون هذا النظام قد صمم بشكل صحيح .والشكل التالي يبين مواصفات لمنزل وهذا الشكل يكون كوسيلة اتصال مابين الشخص الذي يرغب في بناء المنزل (الزبون) وكذلك بين المهندس(المصمم) من جهة وبين المقاول (المنفّذ) الذي سيقوم ببناء المنزل، وفي بناء أنظمة قواعد البيانات يمثل الزبون صاحب النظام ويمثل المصمم (مصمم قاعدة البيانات) والمقاول المنفذ هو مجموعة المبرمجين التي تقوم ببناء النظام .

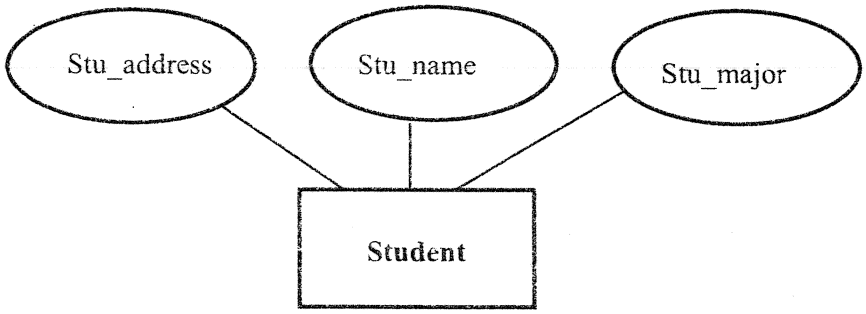
مخطط منزل

نموذج الكيانات والعلاقات:

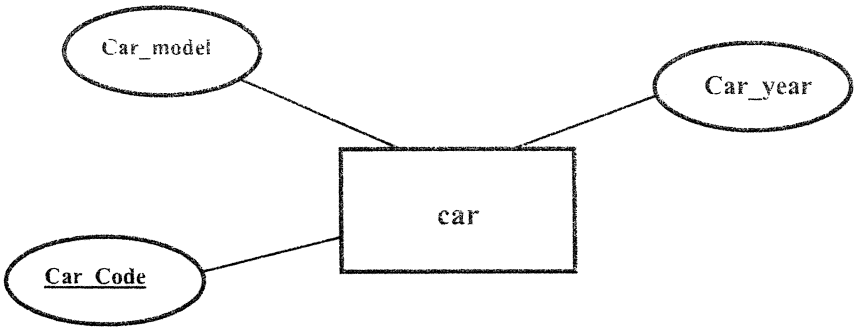
هو عبارة عن نموذج لتمثيل كيانات النظام وصفاتها وكيفية ارتباط هذه الكيانات مع بعضها باستخدام رموز رسومية..ولنتعرف الآن على عناصر هذا النموذج:

مجموعة الكيانات (Entity Set) وتمثل المجموعة التي تنتمي إليها مجموعة الكائنات(Objects) المتشابهة وتمثل بجدول في قاعدة البيانات العلائقية . و الكيان(Entity) هو عبارة عن كائن أو شيء محط الاهتمام في النظام وعلينا أن نقوم بجمع وتسجيل البيانات عن هذا الكيان. مثلا المتدرب ، المقرر، المدرس و الشعبة تعتبر كيانات مهمة في نظام قاعدة البيانات لجامعة .ويمثل الطبيب و المريض و وصفة العلاج كيانات مهمة في قاعدة بيانات لمستشفى . ويرمز لمجموعة الكيانات بمستطيل يحتوي على اسم الكيان .

الخصائص أو الصفات (Attributes): هي عبارة عن الصفات المميزة للكيان، وبعبارة أخرى هي المعلومات الواجب تخزينها عن كائن معين وتمثل بأعمدة الجدول في قاعدة البيانات العلائقية..فمثلا لكل طالب يجب أن نسجل الاسم، الرقم، تاريخ الميلاد، التخصص، ولمنتج معين يكون الرقم الوصف، الطول، العرض، اللون.ويرمز للصفة بشكل بيضاوي يحتوي على اسم الصفة وتربط الصفة مع الكيان بواسطة خط مستقيم.



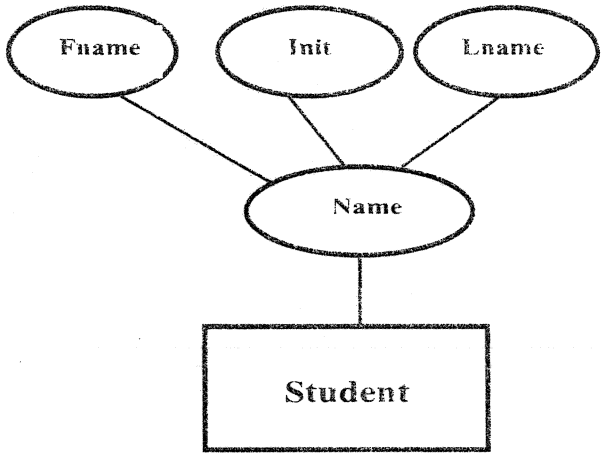
ولكل صفة يجب أن نحدد مجال القيم **(Domain)**: وهو مجموعة القيم لهذه الصفة فمثلا رقم المتدرب يجب أن يكون عدداً صحياً من عشر خانات، واسم المتدرب يجب أن يحتوي على قيم رمزية بطول ٣٠ حرف، والمعدل التراكمي يجب أن يحتوي على عدد كسري مابين ٠.. ٥ مثلا (٣.٥). تاريخ الميلاد يجب أن يكون مقبولاً بحيث لا يتجاوز عمر المتدرب عند القبول ٢٢ سنة. وبعض الصفات يمكن أن تشترك في نفس مجال القيم فمثلا القسم الدراسي للطالب والمدرس يكون اسماً من أسماء الأقسام في الجامعة. والصفة (مجموعة الصفات) التي تم اختيارها كمفتاح رئيس **(primary key)** تُمثل كأي صفة ولكن يوضع خط تحت الاسم.



وفي عملية تحديد الصفات للكيانات لابد من أن نحدد

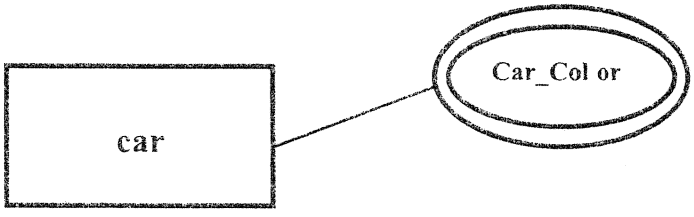
أ - الصفات البسطة والمركبة **Simple and Composite Attributes** :

وتقسم إلى صفات بسيطة أي لا يمكن تجزئتها مثل رقم الطالب، الجنس تاريخ الميلاد. أو مركبة أي يمكن تجزئتها كالاسم (الاسم الأول، الثاني، واسم العائلة)، العنوان (المدينة، الحي، الشارع، رقم المنزل). .ويرمز للصفة المركبة بشكل بيضاوي ترتبط معه أشكال بيضاوية أخرى يحتوي كل منها على اسم الصفة الفرعية وتربط الصفات الفرعية مع الصفة الرئيسة بواسطة خط مستقيم .



ب -صفات وحيدة أو متعددة القيم **Single-Valued or Multiple-Valued Attributes** :

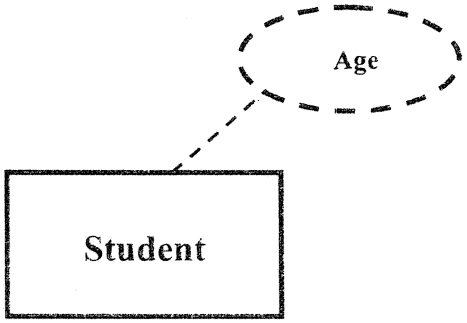
الصفات التي تحتوي على قيمة واحدة مثل (رقم السيارة، تاريخ الصنع) أو عدة قيم مثل لون السيارة (فيمكن أن يكون هناك لون للسقف، الجسم، الجوانب) وكذلك يمكن أن يكون للمدرس أكثر من رقم هاتف أو أكثر من بريد إلكتروني. ويرمز للصفة متعددة القيم بشكل بيضاوي داخل شكل بيضاوي آخر يحتوي على اسم الصفة وتربط الصفة مع الكيان بواسطة خط مستقيم.



ج ـ الصفات المشتقة (**Derived Attributes**):

وهي الصفات التي يمكن اشتقاقها من صفات أخرى ويرمز لها بشكل بيضاوي متقطع يحتوي على اسم الصفة وترتبط مع الكيان بخط مستقيم متقطع أيضا كما في الشكل التالي. مثل عُمر المتدرب يمكن حسابه على أنه الفرق بين تاريخ الميلاد والتاريخ الحالي.

العمر = التاريخ الحالي – تاريخ الميلاد.

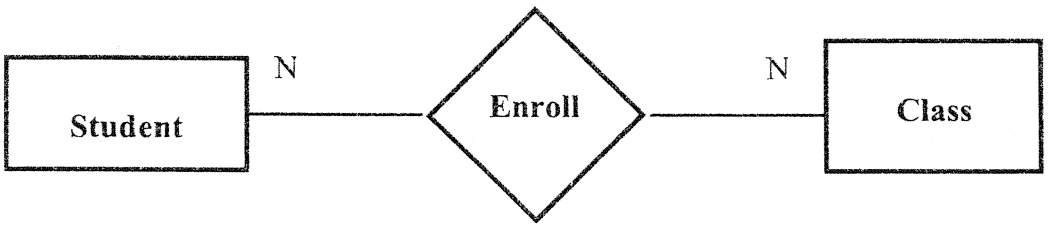


الصفات المشتقة يجب أن لا تخزن ولكن توضع طريقة لحسابها عند عملية الاسترجاع.ولكن قد نخزن بعض الصفات المشتقة إذا كانت عملية حسابها تأخذ وقتا كبيرا وفي نفس الوقت يتم طلبها بشكل كبير مثل المعدل التراكمي للطالب.

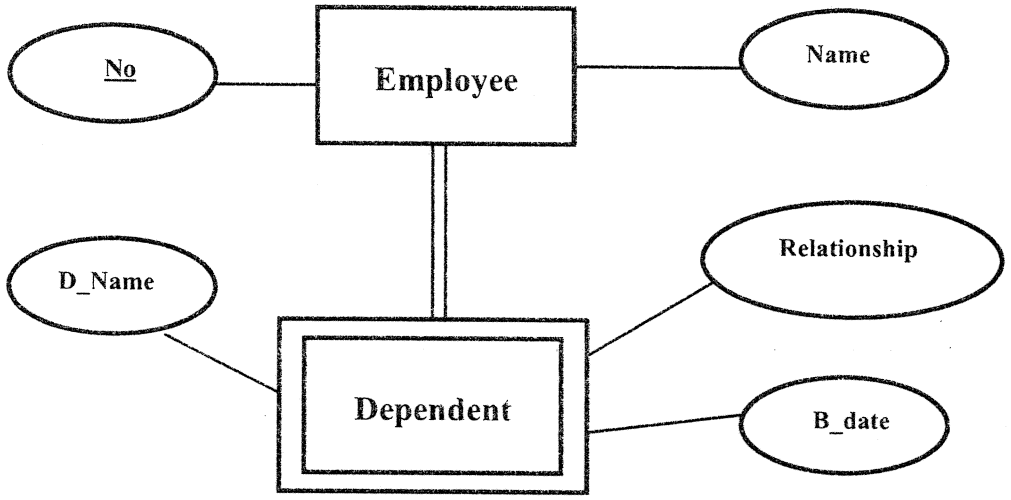
الروابط أو العلاقات (Relationships):

وهي عبارة عن الرابط أو العلاقة مابين الكيانات واسم هذه الرابطة يجب أن يعبر عن كيفية هذا الترابط ويكون على شكل فعل (ينتمي، يحتوي، يسجل، يتكون من....). ويرمز لها بشكل معين يحتوي على اسم الرابط أو العلاقة.وكذلك لكل علاقة درجة تشاركية. وتبين مقدار التشارك مابين الكيانات إما واحد ـ واحد (١:١) أو واحد ـ متعدد(**1: N**) أو متعدد ـ متعدد (**N:N**).

فالطالب يسجل في شعبة أو أكثر والشعبة يسجل فيها مجموعة من الطلاب .



الكيانات الضعيفة: وهي عبارة عن الكيانات التي لا توجد مستقلة بنفسها في النظام وبعبارة أخرى فإن وجودها يعتمد على وجود كيان آخر فمثلا لنفرض أن مؤسسة ما تسجل معلومات عن أسماء الأشخاص التابعين للموظف مثل الأبناء، الزوجة أو الوالدين. فوجود معلومات التابع مرتبط بوجود الموظف وفي هذه الحالة يختار المفتاح الرئيس للكيان الرئيس مع صفة من صفات التابع (مثل الاسم) لتشكل مفتاحا رئيسا للكيان التابع ويوضع تحته خط مقطع. ويرمز للكيان الضعيف بمستطيل داخل مستطيل يحتوي على اسم الكيان الضعيف ويرتبط مع الكيان الرئيس بخطين مستقيمين (يعني أن وجود الكيان الأول شرط لوجود الكيان الآخر وليس بالضرورة للكيانات الضعيفة فقط).

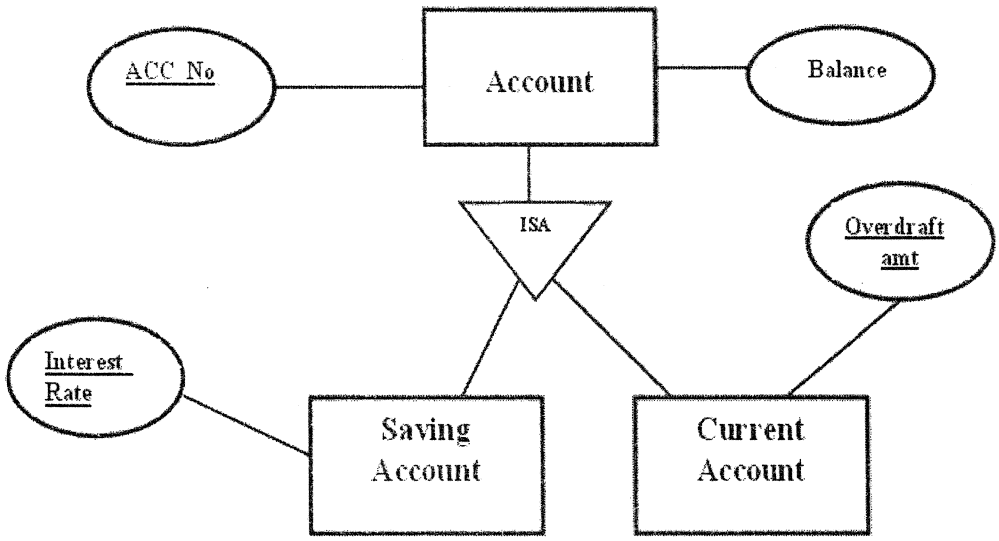


تمثيل الأنواع الرئيسة والأنواع الفرعية (Supertype and Subtype) :

هناك بعض الكيانات الفرعية التي تتبع إلى نوع رئيس (أعلى) Supertype فمثلا بالنسبة للحساب البنكي يمكن أن يكون هناك أكثر من نوع للحسابات ولكن جميع هذه الحسابات تشترك في الكثير من الصفات ففي هذه الحالة نقوم بإنشاء كيان الحساب البنكي Account بحيث يحتوي على جميع هذه الصفات ، ثم بعد ذلك نقوم بإنشاء كيانات فرعية للحسابات يحتوي كل منها على الصفات الخاصة بهذا النوع فقط.

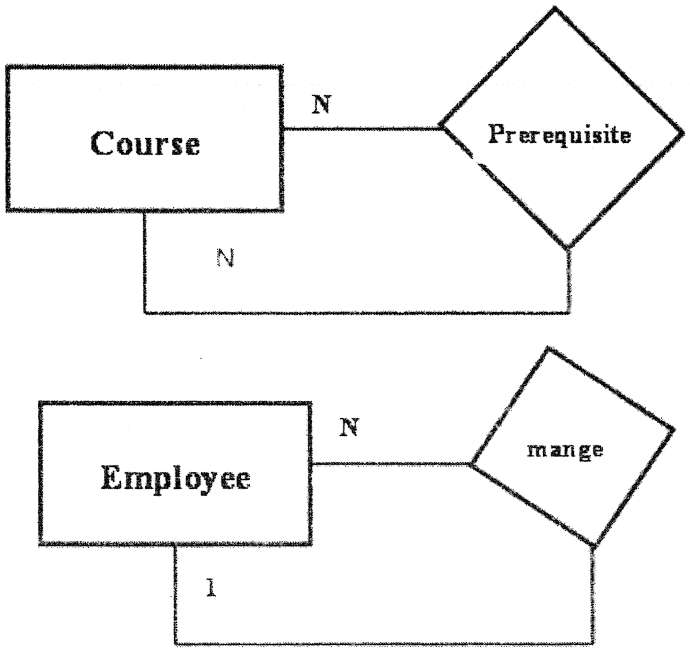
مثال: لنفرض أن لكل الحساب حقل يمثل رقم الحساب وحقل يمثل الرصيد الحالي وفي نفس الوقت لدينا نوعين من الحسابات: الحساب الجاري (Current Account) وفيه الصفة (Overdraft Amount) وهي أعلى قيمة يسمح لصاحب الحساب أن يسحبها عندما لا يكون لديه رصيد. والنوع الثاني حساب التوفير وفية صفة معدل الفائدة(Interest Rate) .

وتمثل العلاقة بين الأنواع الرئيسة العليا والأنواع الفرعية بمثلث مقلوب يحتوي على (ISA) بمعنى يكون.



تمثيل علاقة الكيان مع نفسه (Recursive):

وفي هذه الحالة نبين كيفية تمثيل ارتباط الكيان مع نفسه، فمثلا نفرض أن المقرر الدراسي يمكن أن يكون لديه متطلب سابق أو أكثر (وهذا المتطلب هو عبارة عن مقرر) وكذلك يجب أن يكون للموظف مدير واحد فقط (والمدير بدوره هو أيضا موظف)



حالة دراسية: سنقوم في هذا المثال بعملية تحويل عملية تحليل شركة ما إلى نموذج مفاهيم (نموذج الكيانات و العلاقات ER Diagram). حيث إن الشركة تهتم بتسجيل معلومات عن الأقسام والمشاريع التي تنفذها الشركة وكذلك عن الموظفين العاملين فيها والتابعين لهؤلاء الموظفين .

١ - تقسم الشركة إلى عدة أقسام ولكل قسم اسم واحد ورقم (لا يجوز أن يكون هناك أكثر من قسم بنفس الاسم أو الرقم)، لكل قسم موظف يدير هذا القسم ويجب معرفة التاريخ الذي بدأ فيه هذا الموظف بإدارة القسم ،ولكل قسم موقع واحد فقط.

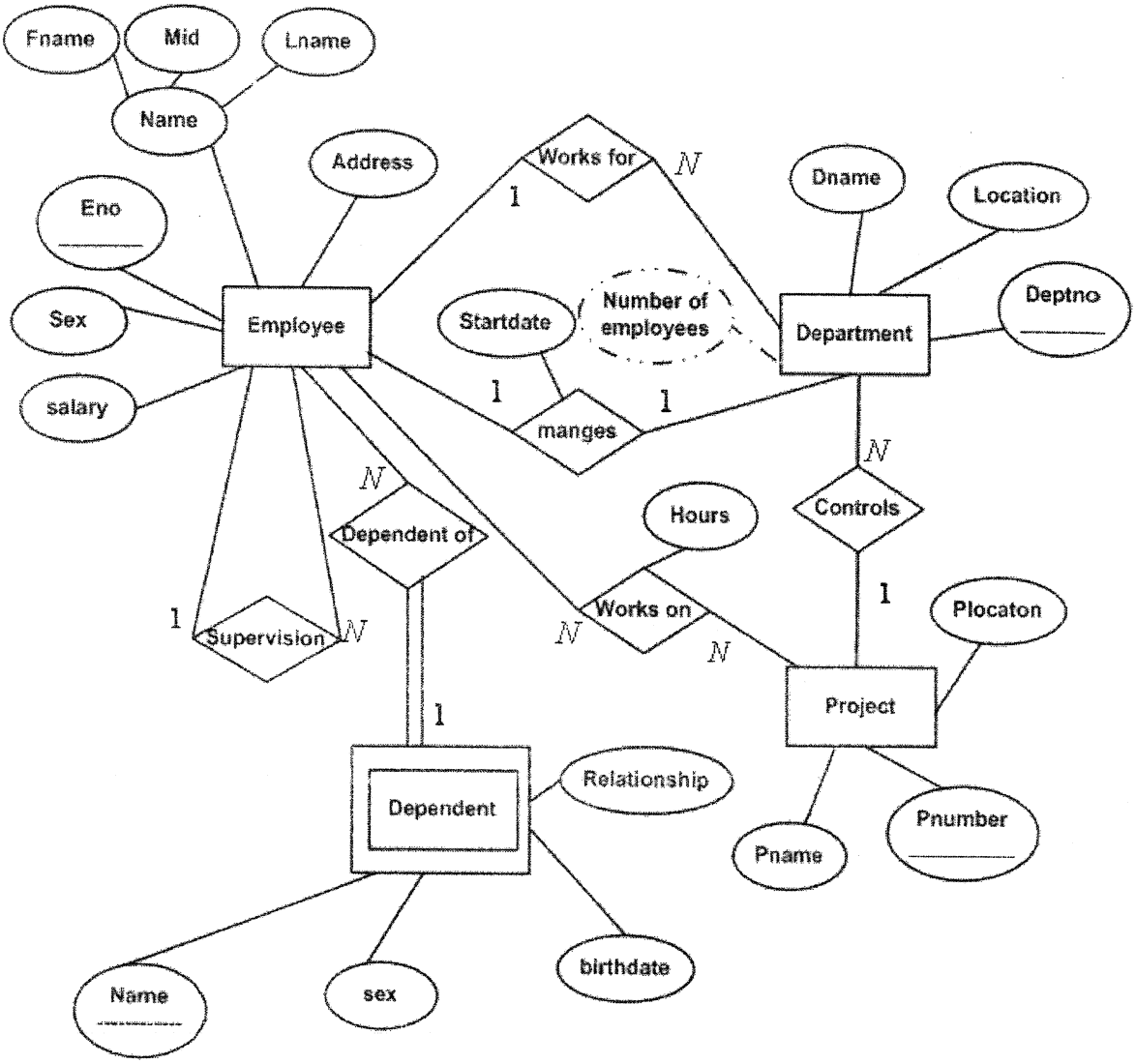
٢ - القسم يمكن أن يدير عدة مشاريع ولكل مشروع رقم واسم ومكان تنفيذ.

٣ - يمكن أن يعمل في القسم موظف أو أكثر ولكن الموظف يجب أن يتبع لقسم واحد فقط ونحتفظ بالمعلومات التالية عن الموظف (الرقم لكل موظف رقم يميزه عن بقية الموظفين، الاسم (الاسم الأول، الاسم الثاني واسم العائلة)، الجنس ، الراتب والعنوان.

٤ - الموظف يمكن أن يعمل في عدة مشاريع (وليس بالضرورة أن يدار المشروع من نفس القسم الذي يتبع إليه الموظف) ونحتفظ بعدد الساعات التي عملها الموظف في كل مشروع.

٥ - تحتفظ الشركة بمعلومات عن التابعين لكل موظف مثل الاسم ،تاريخ ، الميلاد ،الجنس، صلة القرابة .

٦ - تهتم الشركة بمعرفة عدد الموظفين الذين يتبعون لقسم معين.


Fname
Mid
Lname
Name
Address
Works for
N
1
Dname
Location
Eno
Sex
salary
Employee
Startdate
Number of employees
Department
Deptno
manges
1
1
N
N
Dependent of
Hours
Controls
1
Supervision
N
Works on
N
N
Plocaton
Project
1
Dependent
Relationship
Pname
Pnumber
Name
sex
birthdate

الصيغ المعيارية للجداول

مقدمة :

إن عملية وضع تصميم قاعدة البيانات في الصيغة المعيارية يشكل لبنة أساسية في عملية التصميم الجيد لقاعدة البيانات.وتتم هذه العملية على عدة مراحل يتم خلالها تخليص قاعدة البيانات من التكرار غير المسوغ للبيانات بالاعتماد على قوانين الاستنتاج والاعتمادية الوظيفية. وسنقوم في هذا الفصل بالتعرف على الشروط و القوانين اللازمة للوصول بقاعدة البيانات إلى المستوى المعياري الثالث(Third Normal Form **3NF**).

مشاكل تكرار البيانات (Data Anomalies) :

Employee department						
Empno	Ename	Job	Salary	Deptno	Dname	Loc
101	Sami	clerk	3000	10	Accounting	Riyadh
205	Khalid	manager	2500	10	Accounting	Riyadh
303	Ali	salesman	1200	20	Sales	Jeddah
502	Saeed	salesman	2100	20	Sales	Jeddah
601	Salem	clerk	1000	30	Operation	Dmmam

نلاحظ في الجدول السابق أن معلومات الموظف والقسم الذي يعمل فيه موجودة في جدول واحد ونتيجة ذلك تكرار بعض البيانات مثل اسم وموقع القسم في كل سجل وهذا يؤدي إلى عدة مشاكل :

١ - **مشكلة الإضافة** :أي إننا لا نستطيع أن نظيف قسماً جديداً إلا إذا كان القسم يحتوي على، موظف ،لأن المفتاح الرئيس للجدول هو رقم الموظف.

٢ - **مشكلة التعديل** : نلاحظ تكرار اسم وموقع القسم فإذا قمنا بتعديل موقع (Loc)القسم رقم ٢٠ من Jeddah إلى Riyadh فلابد من إجراء عملية التعديل لجميع الموظفين في هذا القسم وإلا ستؤدي هذه العملية إلى عدم توافقية البيانات أي نفس رقم القسم ولكن أكثر من موقع . وكذلك إذا تمت عملية التغيير عند الموظف رقم ٣٠٣ عن طريق الخطأ . وبالتالي لو قمنا بعملية استرجاع لجميع الموظفين الذين يعملون في Jeddah فإن الموظف رقم ٣٠٣ لن يظهر بين الموظفين .

٣ - **مشكلة الحذف** : نلاحظ أن القسم رقم ٣٠ يحتوي على موظف واحد فقط ،فلو قمنا بحذف الموظف رقم ٦٠١ فإن معلومات القسم رقم ٣٠ سوف تختفي من الجدول .

الاعتمادية الوظيفية (Functional Dependency FD):

وهي اعتماد قيمة إحدى صفات الكيان على قيمة صفة(صفات) أخرى ويرمز لها بالرمز (⟵)

مثال $A \longrightarrow B$

يعني أن B تعتمد اعتمادا وظيفيا على A وهنا نستطيع أن نقول أن قيمة A تحدد قيمة B . ومن خلال تحديد الاعتمادية نستطيع أن نحدد المكان الذي يجب أن توضع فيه الصفة وهذا بالتالي يؤدي إلى وضع البيانات في المكان الصحيح ونتخلص من عملية تكرار البيانات وما يترتب على التكرار من مشاكل(Anomalies) .

مثال: لكل موظف اسم واحد فقط ولكل موظف قسم واحد يعمل فيه إذا :

FD1 : Empno ⟶ Ename

FD2 : Empno ⟶ Deptno

ويمكن أن نعيد كتابة هذه الاعتمادية على الشكل التالي

FD1 : Empno ⟶ Ename,Deptno

FD :Functional Dependency

قواعد الاستنتاج

وهي عبارة عن مجموعة من القواعد تستخدم في عملية تحديد الاعتمادية الوظيفية (Functional Dependency FD) وتتلخص هذه القواعد بستة قواعد على النحو التالي :

١ - الانعكاسية **Reflexive** :إذا كانت Y جزء من X ((Y محتواه في X)

فإن X تحدد Y

1- $X \supseteq Y : \quad X \rightarrow Y$

٢ - قاعدة الزيادة أو الإضافة **Augmentation** :إذا كان X تحدد Y فإن إضافة Z إلى X تعني أنه بالإمكان إضافة Z إلى Y

2- $\{X \rightarrow Y\} \models XZ \rightarrow YZ$

٣ - قاعدة التعدي **Transitive** : تعني أنه إذا كانت X تحدد Y وكانت Y تحدد Z فإن X تحدد Z .

3- $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$

٤ - قاعدة الاتحاد **Union** : تعني أنه إذا كانت X تحدد Y و X تحدد Z فإننا نستطيع أن نقول أن X تحدد YZ .

4- $\{X \rightarrow Y, X \rightarrow Z\} \models X \rightarrow YZ$

٥ - قاعدة التقسيم Decomposition وهي عكس قاعدة الاتحاد

5- $\{X \rightarrow YZ\} \models X \rightarrow Y, X \rightarrow Z$

٦ - قاعدة التعدي الزائف **Pseudotransitive** تشبه قاعدة التعدي

6- $\{X \rightarrow Y, WY \rightarrow Z\} \models WX \rightarrow Z$

$\models$ تعني أنه إذا تحقق الطرف الأيسر فإننا نستطيع استنتاج الطرف الأيمن .

تعريف الصيغة المعيارية الأولى(First Normal Form 1NF):

نستطيع أن نقول أن الجدول في الصيغة المعيارية الأولى إذا كانت جميع أعمدة الجدول تحتوي على بيانات بسيطة أو مفردة(غير مركبة) أي إن كل عمود يحتوي على قيمة واحدة فقط .

مثال ١ يمثل الجدول التالي معلومات موظف Employee:

No	Name			Adresse		
	Fname	Mid	Lname	city	Street	House no
100	Ali	Salem	musa	Riyadh	Immam saud	210
120	Saeed	Eisa	Ali	Riyadh	King Fahad	202

نلاحظ في الجدول أن الاسم يتكون من ثلاثة أجزاء وكذلك العنوان فبالتالي لا نستطيع أن نخزن قيمة واحدة في عمود الاسم وكذلك بالنسبة للعنوان وهذا يخالف شروط قاعدة البيانات بأن العمود يجب أن يحتوي على قيمة واحدة فقط.وهذا يعني أن الجدول السابق لا ينطبق عليه شرط الصيغة المعيارية الأولى **1NF,** ولوضع الجدول في الصيغة المعيارية الأول يجب تقسيم الأعمدة المركبة إلى أعمدة بسيطة

No	Fname	Mid	Lname	city	Street	House no
100	Ali	Salem	musa	Riyadh	Immam saud	210
120	Saeed	Eisa	Ali	Riyadh	King Fahad	202

لقد قمنا بتقسيم الأعمدة المركبة إلى أعمدة بسيطة وبالتالي نستطيع أن نقول أن الجدول الآن في الصيغة المعيارية الأولى **1NF** .

مثال ٢ : يمثل الجدول التالي سجل ساعات العمل HOURS لموظف في عدد من المشاريع PROJECTS والقسم الذي يشرف على تنفيذ المشروع

NO	Name	Project_Code	Hours	Deptno	Dname
210	Ali	P1,p2,p3	12,20,40	10,20,20	Research, Operation, Operation
201	Salem	P1,p3	30,15	10,20	Research Operation
305	Ali	P2,p3	40,20	20,20	Operation; Operation

كما هو مبين في الجدول السابق فإن هناك عدداً من الأعمدة تحتوي على أكثر من قيمة مثل رمز المشروع Project_Code وكذلك عدد ساعات العمل Hours والأقسام Deptno التي تشرف على المشاريع . وهذا يعني أن الجدول ليس في الصيغة المعيارية الأولى، ولتحويله يجب أن نقوم بتقسيم الجدول على النحو التالي للتخلص من هذه المشكلة.

NO	Name	Project_Code	Hours	Deptno	Dname
210	Ali	P1	12	10	Research
210	Ali	p2	20	20	Operation
210	Ali	p3	40	20	Operation
201	salem	P1	30	10	Research
201	salem	p3	15	20	Operation
305	Ali	P2	40	20	Operation
305	Ali	p3	20	20	Operation

ولكن تبرز هنا لدينا مشكلة جديدة وهي إيجاد مفتاح رئيسي للجدول إذ أصبح رقم الموظف لا يصلح لأن يكون مفتاحا رئيسا للجدول (Primary Key) لأن من شروط المفتاح الرئيس أن لا يتكرر في أكثر من صف .لنقوم الآن باستخدام الاعتمادية الوظيفية لمحاولة إيجاد المفتاح الرئيس للجدول

FD 1 :No → Name

حيث إن لكل موظف اسم واحد .

FD 2 : Project_Code→ Deptno

حيث إن لكل مشروع قسم واحد يشرف عليه .

FD 3 : Deptno →Dname

حيث إن لكل قسم اسم واحد.

أما بالنسبة لبقية العناصر فمثلا اسم الموظف لا يحدد شيئاً لأنه يوجد هناك أكثر من موظف اسمه Ali فالاسم لا يحدد الرقم وكذلك فإن علي يعمل في أكثر من مشروع .

وكذلك رمز لمشروع لا يحدد عدد الساعات ولاالموظفين الذين يعملون فيه فالمشرع P1 يعمل فيه أكثر من موظف وبساعات مختلفة .

أما بالنسبة للقسم فلا يحدد الموظفين ولا المشاريع فمثلا القسم ٢٠ يشرف على أكثر من مشروع هذه المشاريع يعمل فيها أكثر من موظف .

ففي هذه الحالة يجب علينا القيام بمحاولة جديدة لإيجاد المفتاح الرئيس وتتلخص هذه العملية بمحاولة إيجاد مفتاح مركب(تركيب أكثر من صفة لتشكل المفتاح الرئيس) يقوم بتحديد جميع الصفات في الجدول :

سنقوم بأخذ رقم الموظف مع رقم المشروع

FD 4 :No, Project_Code→ name
FD 5 :No, Project_Code→ Deptno
FD 6 :No, Project_Code→ Hours
FD 7 : Deptno →Dname
FD 8 :No, Project_Code→ Name ,Hours, Deptno, Dname

FD4,FD5 تنطبق من FD1,FD2 حيث إن رقم الموظف وحدة يحدد الاسم وكذلك رمز المشروع يحدد القسم ، أما بالنسبة ل FD5 فإنها تنطبق لأن رقم الموظف ورمز المشروع يحددان عمل الموظف في ذلك المشروع ، وبالتالي نكون قد حصلنا على مفتاح رئيس لهذا الجدول وكذلك قمنا بوضعه في الصيغة المعيارية الأولى (**1NF**).

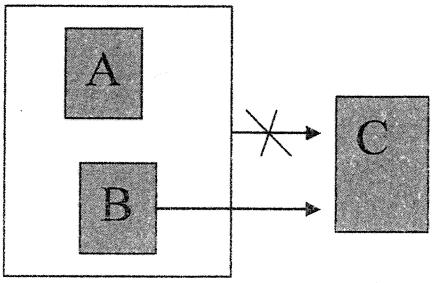
تعريف الصيغة المعيارية الثانية(Second Normal Form 2NF):

نستطيع أن نقول أن الجدول في الصيغة المعيارية الثانية:

١ - إذا كان الجدول في الصيغة المعيارية الأولى.

٢ - إذا لم يحتوي الجدول على اعتمادية جزئية.

الاعتمادية الجزئية: هي أن تعتمد بعض الأعمدة (الصفات) اعتمادا وظيفيا على جزء من المفتاح الرئيس



نلاحظ أن A,B تحدد C أي إن C تعتمد اعتمادا وظيفيا على A,B وكذلك أن B تحدد C أي إن C تعتمد اعتمادا وظيفيا B. وفي هذه الحالة نستطيع أن نقول أن هذا الجدول يحتوي على اعتمادية جزئية .

NO	Name	Project_Code	Hours	Deptno	Dname
210	Ali	P1	12	10	Research
210	Ali	p2	20	20	Operation
210	Ali	p3	40	20	Operation
201	Salem	P1	30	10	Research
201	Salem	p3	15	20	Operation
305	Ali	P2	40	20	Operation
305	Ali	p3	20	20	Operation

والآن هل الجدول السابق في الصيغة المعيارية الثانية ؟

وللإجابة على ذلك نجيب على السؤالين التاليين :

١ - هل الجدول في الصيغة المعيارية الأولى ؟

نعم، لأنه لا توجد هناك قيم متكررة ،كل عمود يحتوي على قيمة واحدة فقط .

٢ - هل توجد هناك اعتمادية جزئية ؟

ولمعرفة ذلك يجب أن نحدد الاعتمادية الوظيفية

FD 1 :No → Name

FD 2 : Project_Code→ Deptno,Dname

FD 3 :No, Project_Code→ name ,deptno, hours

المفتاح الرئيس هو No, Project_Code ولكن No يحدد Name إذا هناك اعتمادية جزئية وكذلك

Project_Code يحدد deptno وDname و هذه اعتمادية جزئية أخرى . وللتخلص من هذه المشكلة يجب أن نقوم بتقسيم الجدول إلى جداول بحيث يضم كل منها الجزء من المفتاح والأعمدة التي تعتمد عليه ونبقي فقط المفتاح المركب مع الأعمدة التي تعتمد عليه:

١ - نقوم بنقل اسم ورقم الموظف إلى جدول جديد ونبقي نسخة من رقم الموظف في الجدول الأصلي (لأنه جزء من المفتاح الرئيس) .

٢ - نقوم بنقل رمز المشروع ورقم القسم إلى جدول جديد ونبقي نسخة رمز المشروع في الجدول الأصلي (لأنه جزء من المفتاح الرئيس) .

٣ - نبقي بقية الأعمدة كما هي (عدد الساعات).

٤ - وبالتالي تصبح الجداول على النحو التالي بعد عملية التقسيم :

NO	Project_Code	Hours
210	P1	12
210	p2	20
210	p3	40
201	P1	30
201	p3	15
305	P2	40
305	p3	20

NO	Name
210	Ali
210	Ali
210	Ali
201	salem
201	salem
305	Ali
305	Ali

Project_Code	Deptno	Dname
P1	10	Research
p2	20	Operation
p3	20	Operation

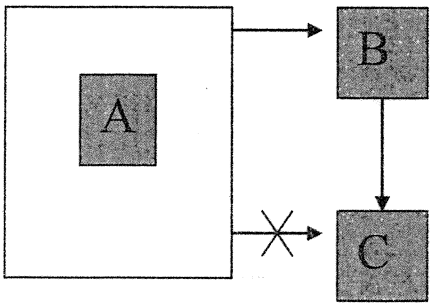
تعريف الصيغة المعيارية الثالثة(Third Normal Form 3NF):

نستطيع أن نقول أن الجدول في الصيغة المعيارية الثالثة:

١ - إذا كان الجدول فيالصيغة المعيارية الثانية.

٢ - إذا لم يحتوي الجدول على اعتمادية متعدية.

الاعتمادية المتعدية: هي أن تعتمد بعض الأعمدة (الصفات) اعتمادا وظيفيا على صفة غير المفتاح الرئيس.



نلاحظ أن A تحدد B, C أي إنB, C تعتمد اعتمادا وظيفيا على A وكذلك أن B تحدد C أي إن C تعتمد اعتمادا وظيفيا B. وفي حالة نستطيع أن نقول أن هذا الجدول يحتوي على اعتمادية متعدية.

والآن هل الجداول السابقة في الصيغة المعيارية الثالثة ؟

وللإجابة على ذلك نجيب على السؤالين التالين:

١ - هل الجداول في الصيغة المعيارية الثانية ؟

نلاحظ أن جميع الجداول في الصيغة المعيارية الثانية حيث لا يوجد فيها اعتمادية جزئية .

٢ - هل توجد هناك اعتمادية متعدية ؟

ولمعرفة ذلك يجب أن نحدد الاعتمادية الوظيفية لكل جدول

أ - الجدول الأول

FD 1 :No → Name

لا توجد اعتمادية متعدية .

ب - الجدول الثاني

FD 1 :No, Project_Code→ hours

لا توجد اعتمادية متعدية.

ج - الجدول الثالث

FD 1 : Project_Code→ Deptno,Dname

FD 2 : Deptno → Dname

المفتاح الرئيس هو Project_Code يحدد Dname وDeptno وفي نفس الوقت فإن وDeptno يحدد Dname أي إن هناك اعتمادية متعدية . وللتخلص من هذه المشكلة يجب أن نقوم بتقسيم الجدول إلى جداول بحيث يضم كل منها الأعمدة التي تعتمد على بعض ونبقي المفتاح مع الأعمدة التي تعتمد عليه وحدة فقط مع إبقاء المحدد الجديد (Deptno)

١ - نقوم بنقل رقم و اسم القسم إلى جدول جديد ونبقي نسخة من رقم القسم في الجدول الأصلي.

٢ - وبالتالي تصبح الجداول على النحو التالي بعد عملية التقسيم :

NO	Project_Code	Hours
210	P1	12
210	p2	20
210	p3	40
201	P1	30
201	p3	15
305	P2	40
305	p3	20

NO	Name
210	Ali
210	Ali
210	Ali
201	Salem
201	Salem
305	Ali
305	Ali

Project_Code	Deptno

Deptno	Dname

P1	10
p2	20
p3	20

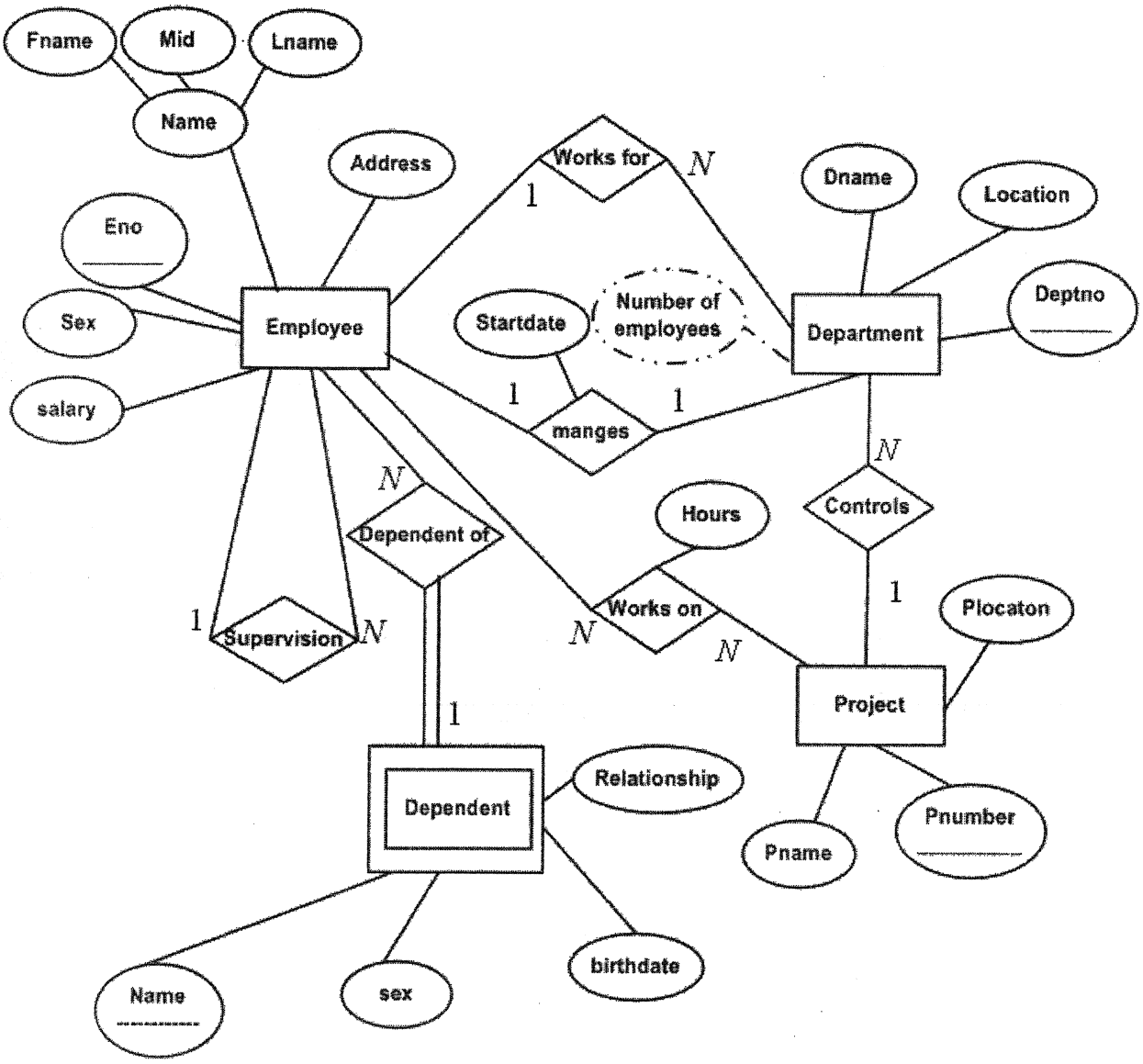
10	Research
20	Operation

الآن نستطيع أن نقول أن هذه الجداول هي في الصيغة المعيارية الثالثة **3NF** وتعتبر هذه الصيغة مقبولة لمعظم مصممي قواعد البيانات .

تحويل نموذج الكيانات والعلاقات إلى نموذج علائقي

مقدمة

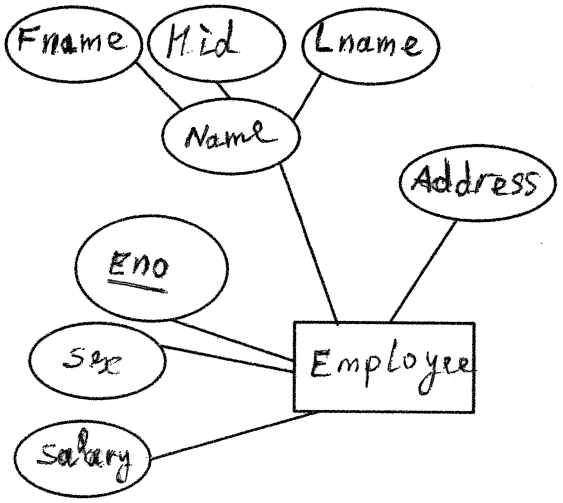
لتحويل عملية التصميم إلى قاعدة بيانات لابد في البداية من تحويل نموذج الكيانات والعلاقات) إلى نموذج علائقي حتى نسهل عملية تنفيذ هذا النموذج في قاعدة (إنشاء الجداول).وسنقوم في هذا الفصل بدراسة كيفية تحول نموذج المفاهيم(نموذج الكيانات والعلاقات) إلى نموذج علائقي مستخدمين المثال السابق للشركة .



والآن سنقوم بعدة خطوات لتحويل نموذج المفاهيم(نموذج الكيانات والعلاقات) إلى نموذج علائقي :

تحويل لكيانات :

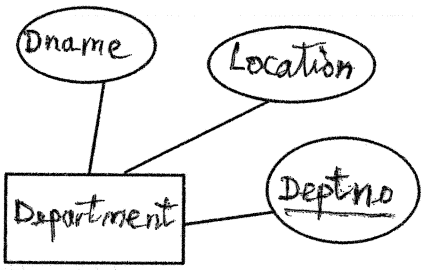
١ - لكل كيان(E) Entity في النموذج قم بإنشاء علاقة Relation (R) بحيث تحتوي العلاقة على جميع الصفات البسيطة غير المركبة وإذا كانت الصفات مركبة قم بتقسيمها إلى صفات بسيطة ،ثم قم باختيار صفة أو أكثر لتشكل المفتاح الرئيس للعلاقة .



نقوم بتحويلها لتصبح على الشكل التالي:

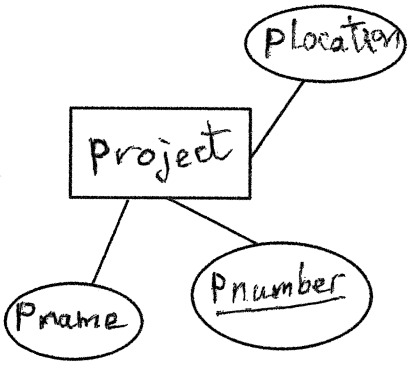
Employee						
Eno	Fname	Mid	Lname	sex	Address	Salary

لاحظ أننا قمنا بتقسيم الاسم (صفة مركبة) إلى مكونات بسيطة .



Department		
Deptno	Dname	Location

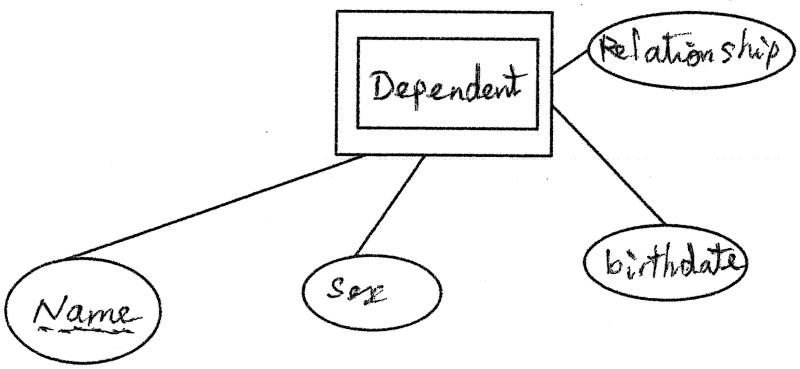
لاحظ أننا لم نقم بإضافة عدد الموظفين (صفة مشتقة) ولكن يجب أن تأخذ بعين الاعتبار لإيجاد عدد الموظفين عن طريق بناء آلية استرجاع (Query).



Project		
Pnumber	Pname	Plocation

تحويل لكيانات الضعيفة Weak Entity :

لكل كيان ضعيف (Weak Entity) في النموذج قم بإنشاء علاقة Relation (R) بحيث تحتوي العلاقة على جميع الصفات البسيطة غير المركبة وإذا كانت الصفات مركبة قم بتقسيمها إلى صفات بسيطة ،ثم قم باختيار إحدى الصفات مع المفتاح الرئيس للكيان الذي يتبع إليه الكيان الضعيف لتشكل المفتاح الرئيس للكيان،ثم قم بإنشاء مفتاح أجنبي ليشير إلى الكيان الذي يتبع الكيان الضعيف (المفتاح الرئيس لذلك الكيان).

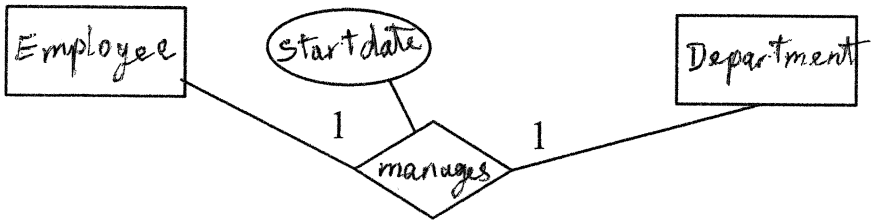


Dependent				
Eno	Name	Sex	Birthdate	Relationship

تحويل التشاركية: كما مر معنا سابقا فهناك ثلاثة أنواع من التشاركية علاقة واحد ـ واحد (١:١) وعلاقة واحد ـ متعدد (١:N) علاقة متعدد ـ متعدد (N:N) وسنقوم بعملية التحويل كل منها على النحو التالي :

١ - علاقة واحد ـ واحد (١:١)

لكل علاقة واحد ـ واحد (١:١) قم باختيار أحد الكيانين لتحتوي على مفتاح أجنبي ليشير إلى الكيان الآخر .

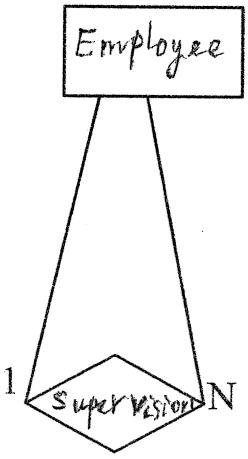


ففي هذه الحالة نقوم بإضافة صفة جديدة (Mgr) لتشير إلى الموظف الذي يتولى إدارة القسم (مفتاح أجنبي لجدول الموظفين) وكذلك إضافة تاريخ بداية إدارة هذا الموظف لذلك القسم .

Department				
Deptno	Dname	Location	Mgr	Startdate

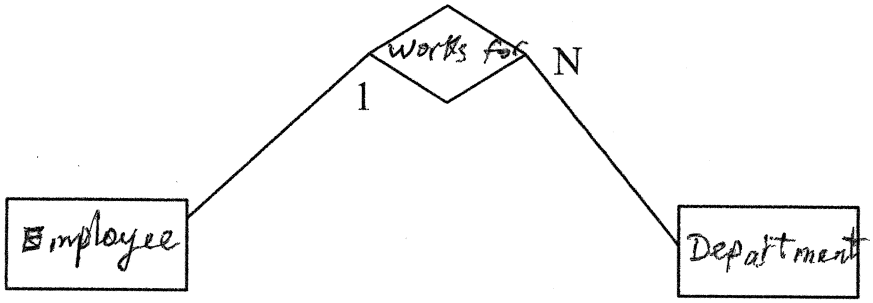
٢ - علاقة واحد ـ متعدد (1 :N)

لكل علاقة واحد ـ متعدد(N:1) قم بإضافة عمود(أعمدة) لتكون مفتاحا أجنبيا في جانب المتعدد(N) ليشير إلى المفتاح الرئيس في جانب الواحد(١) .



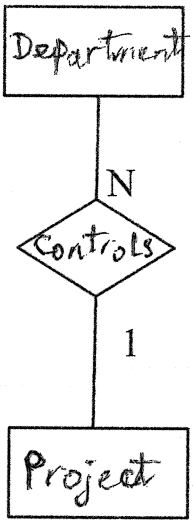
Employee							
Eno	Fname	Mid	Lname	sex	Address	Salary	Mgr

وفي هذه الحالة نقوم بإضافة صفة جديدة (Mgr) لتشير إلى الموظف الذي يتولى الإشراف على الموظف (مفتاح أجنبي لنفس الجدول)



Employee								
Eno	Fname	Mid	Lname	sex	Address	Salary	Mgr	Deptno

وفي هذه الحالة نقوم بإضافة صفة جديدة (Deptno) لتشير إلى القسم الذي يتبع إليه الموظف (مفتاح أجنبي لجدول الأقسام)

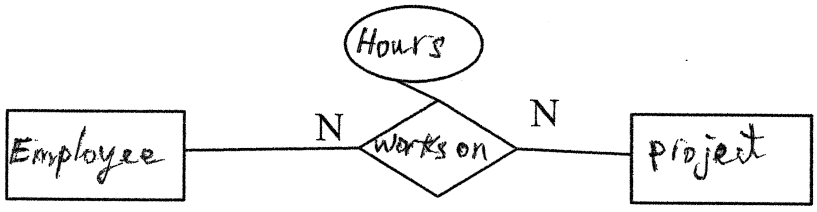


Project			
Pnumber	Pname	Plocation	Deptno

وفي هذه الحالة نقوم بإضافة صفة جديدة (Deptno) لتشير إلى القسم الذي يدير هذا المشروع (مفتاح أجنبي لجدول الأقسام).

٣ -علاقة متعدد ـ متعدد (N:N)

لكل علاقة متعدد ـ متعدد (N:N) قم بإنشاء علاقة جديد يكون المفتاح الرئيس لها عبارة عن دمج المفاتيح الرئيسة في طرفي العلاقة.وإضافة أي صفات جديد لهذه العلاقة



Works for		
Eno	Pnumber	Hours

ففي هذه الحالة نقوم بإنشاء جدول جديد يحتوي (رمز المشروع،رقم الموظف ،عدد ساعات العمل) بحيث يشكل (رمز المشروع،رقم الموظف) المفتاح الرئيس للجدول وبنفس الوقت يكون رمز المشروع مفتاحاً أجنبياً لجدول المشاريع ، و رقم الموظف مفتاحاً أجنبياً لجدول الموظفين .

تحويل العلاقة بين الأنواع الفرعية(Subtype) والأنواع العليا(ISA (Super Type

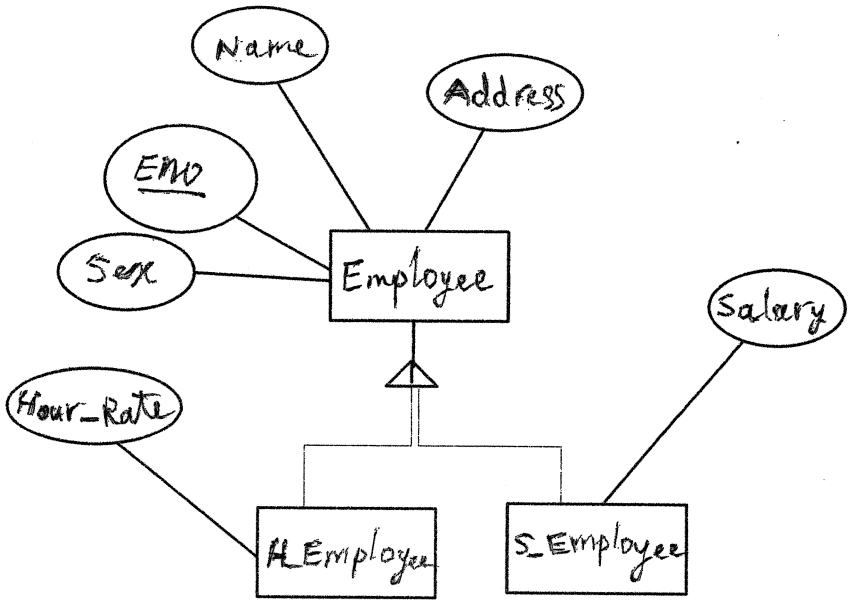
وذلك عن طريق وضع المفتاح الرئيس في النوع الفرعي ليكون مفتاحا رئيسيا لهذا الجدول وفي نفس الوقت يكون مفتاحا أجنبيا للنوع الأعلى:

لنفرض أن لدينا نوعين من الموظفين

١ - موظف دائم يكون له راتب ثابت

٢ - موظف يعمل بالساعة ونسجل له أجرة العمل عن كل ساعة

فبالتالي يكون النموذج على الشكل التالي .



فناتج عملية التحويل يكون على النحو التالي:

H_Employee	
Eno	Hour_Rate

S_Employee	
Eno	Salary

الجبر العلائقي
(Relational Algebra)

الجبر العلائقي أساس لغات استعلام قاعدة البيانات ويستعمل نفس الأسس التي يستند إليها الجبر الحسابي ، والعلاقات (الجداول) هي معاملات مجموعة عمليات الجبر العلائقي ويجب ان يكون لها نفس المخطط والناتج.

الجبر العلائقي عملياته لا تنفذ بواسطة الحاسوب وهو ليس لغة برمجة (ملفات استعلام علائقي) ويطبق على العلاقات لحظياً وبشكل مؤقت ويستخدم لمعالجة العلاقات والناتج يكون علاقة مؤقتة أيضاً بدون التأثير على محتوي العلاقة أو العلاقات الأصلية ولكن السجلات تتأثر بدون تكرار

رموز عمليات الجبر العلائقي ليست بالضرورة نفس رموز عمليات SQL حتى لو لهن نفس الاسم وهناك ستة عمليات أساسية هي:

Rname,union,project,select,Cartesian product,set difference

يتم التعبير عن الاستعلامات بالعلاقات الجبرية، الاستعلامات تعابير في الجبر العلائقي.

ويمكن تلخيص البناء الأساسي للجبر العلائقي بما يلي:

1- علاقات.
2- عمليات تجرى على العلاقات.
3- لا تتأثر العلاقات الأصلية.
4- والنتائج عبارة عن علاقات جديدة.

هناك مجموعتان من الأدوات:

1- مجموعة رياضية نظرية على أساس العلاقات مثل عمليات الاتحاد والتقاطع والفرق والضرب الكارتيزي (الديكارتي)
2- مجموعة عمليات خاصة بقواعد البيانات مثل الاختيار ،والعرض ، والضم.

اسم العملية	رمز العملية	الناتج	
عملية الإتحاد union	U	R U S ➔ T	إذا كانت العلاقتان R,S تحتوي على Iوj من الصفوف على التوالي فإن ناتج اتحاد العلاقتان يكون علاقة جديدة مثلا T تحتوي على الأكثر (I+J) صف دون تكرار أي العلاقة الجديدة تحتوي كل السجلات في العلاقة R والعلاقة S أو في العلاقتان معا، وإزالة السجلات المكررة. وهي تمثل عملية المنطق (أو OR) وهي عملية تبادلية R U S ↔S U R R U (S U T)=(R U S) U T توزيعية الدرجة يجب أن تتوافق مثال: R U S ➔ T A 49 B 54 D 64 Table R A 49 B 54 table S D 64 A 49
عملية التقاطع intersection	∩	R ∩ S ➔ T	إذا كانت العلاقتان R,S تحتوي على Iوj من الصفوف على التوالي فإن تقاطع العلاقتان يكون علاقة جديدة مثلا T تحتوي على الصفوف الموجودة (المكررة) في كلا العلاقتين فقط R,S وهي تماثل عملية المنطق(و and) وهي عملية تبادلية R ∩S= S ∩R R ∩(S ∩T)=(R ∩S) ∩T توزيعية الدرجة يجب أن تتوافق table T= R ∩S A 49 Table R A 49 B 54 table S D 64 A 49
عملية الطرح (الفرق) difference	-	R – S ➔ T	إذا كانت العلاقتان R,S تحتوي على Iوj من الصفوف على التوالي فإن طرح العلاقتان على الأكثر R-S صف تكون علاقة جديدة مثلا T، تحتوي على الصفوف(السجلات) الموجودة في R وغير موجودة في العلاقة S وهي عملية غير تبادلية R-S ≠ S-R الدرجة يجب أن تتوافق

عملية الضرب الكارتيزي cartesianproduct	×	R × S ➔ T

إذا كانت العلاقتان R,S تحتوي على J,I من الخصائص على التوالي و n,m من الحقول(أعمدة) على التوالي فإن الناتج ضرب العلاقتان يكون علاقة جديدة مثلا T تحتوي على الأكثر (i*j) صف و(m+n) عمود، وتشمل كل الأزواج الممكنة للعلاقتان R2,R1.

الدرجة للنتيجة= مجموع الدرجات للمعاملات.

R(I1,I2,I3,I4,…,In)×S(J1,J2,J3,J4,…,Jm)= T(I1,I2,…In,J1,J2,…Jm)

حيث سجلات T تمثل سجل لكل جمع بين سجلات S,R و n,m خصائص وليس لها معنى إلا إذا استخدمت مع غيرها من العمليات

Table R

A	49
B	54

table S

D	64
A	46

table T=R×S

A	49	D	64
A	49	A	49
B	54	D	64
B	54	A	49

عملية الانتقاء select	σ	$\sigma_{<condihon)}$ (R)

إذا كانت العلاقة R تحتوي على عدد من الخصائص (A,B,C,D,….,K) R على التوالي وعلى عدد من الصفوف ($I_1,I_2,….I_n$) فان ناتج عملية الانتقاء يكون علاقة تحتوي على كل خصائص العلاقة ولكن يعرض الصفوف التي تحقق شرط ما في علاقة ما.

إذاً تستخدم عند الحاجة لانتقاء مجموعة جزئية من الصفوف الأفقية من علاقة والتي تحقق الشرط الانتقاء في أمر الانتقاء ، ويتم حذف ما تبقى من السجلات (الصفوف) التي لا تحقق الشرط دون التأثير على الخصائص.

ويتم حذف السجلات المكررة من العلاقة

امثلة:

بناءً على العلاقة التالية S

sid	S_name	rating	age
28	Yunes	9	35.0
31	Lubna	8	55.5
44	Jamal	5	35.0
58	Hamza	10	35.0

ما ناتج العمليات (الأوامر) التالية:

1) $\sigma_{rating>8}(S)$

2) $\sigma_{s\ name\ =\ "gamal"}(S)$

الناتج

1)

sid	S_name	rating	Age
25	Yunes	9	35.0
58	Hamza	10	35.0

2)

sid	S_name	rating	Age
44	Jamal	5	35.0

Table R

a	49
b	54

table S

d	64
a	49

table T ➔ R-S

b	54

عملية الإسقاط project	π	π/pi π<attribute list>(R)	ملاحظة: يمكن تداخل عمليات الإسقاط والانتقاء بامر واحد وقد يكون ناتج علاقة مدخل العلاقة جبرية جديدة (عملية مرتبة). $\pi_{sname,rating}(\sigma_{rating>8}(S_2))$

S_name	Rating
Yunes	9
Hamza	10

للوصول على قائمة من ارقام الموظفين الذين يعملون في دائرة رقم 1.

$\pi_{empnum}(\sigma_{deptno=1}(employee))$

يكتب الأمر بلغة SQL كما يلي:

SQL

select empnum from employee wherendeptnum=1;

إذا كانت العلاقة R تحتوي على عدد من الخصائص $R(n_1,n_2,n_3 \ldots n_n)$ على التوالي فإن ناتج عملية الإسقاط يكون علاقة تحتوي على الخصائص المحددة في قائمة الخصائص ، وتظهر الخصائص في العلاقة بنفس ترتيب الخصائص في قائمة الخصائص.

إذاً تستخدم عند الحاجة لانتقاء مجموعه جزئية من الخصائص (أعمده الجدول الكلية) من علاقة بتحديد أسماء الخصائص المطلوبة في قائمة الخصائص ويتم حذف ما تبقى من الخصائص دون التأثير على الصفوف وربما يتم اختزال الصفوف المكررة من العلاقة.

درجة العلاقة = عدد الخصائص في القائمة (العلاقة) العلاقة الجديدة درجتها = نفس درجة قائمة الخصائص بدون تكرار للسجلات.

امثلة: بناءً على العلاة التالية S.

sid	S_name	rating	age
28	Yunes	9	35.0
31	Lubna	8	55.5
44	Jamal	5	35.0
58	Hamza	10	35.0

ما ناتج الاوامر (العمليات) التالية:

1) $\pi_{S\_name,age}(S)$ 2) $\pi_{rating}(S)$

(1

S_name	Age
Yunes	35.0
Lubna	55.5
Jamal	35.0
Hamza	35.0

(2

Rating
9
8
5
10

مراحل تطور لغة SQL

مرت اللغة بمراحل التطور التالية منذ نشأتها عام 1970:

المعيار الذي سنقوم باستخدامه في مادتنا هو المعيار SQL 99.

انتبه:

معيارية ولكن!!!:

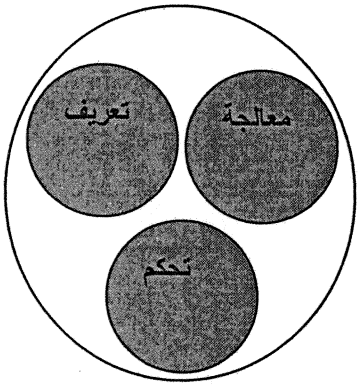
بالرغم من كون SQL لغة معيارية فقد تم اعتمادها من قبل شركات تطوير أنظمة إدارة قواعد البيانات بأشكال مختلفة وتم إقحام بعض الإضافات أو تعديل بعض التعليمات. على كل حال، بقيت اللغة في غالبها تدعم التعليمات الرئيسية.

سننوه إلى بعض هذه الاختلافات في بعض الأمثلة التطبيقية التي نقدمها.

SQL

تعمل SQL بمبدأ توجيه طلب إلى محرك قاعدة البيانات والحصول على جواب منه بحيث نحصل على مجموعة نتائج.
توفر SQL مجموعة من التعليمات بحيث يمكن تقسيمها إلى ثلاث لغات فرعية:

- لغة معالجة البيانات: Select, insert, delete, update
- لغة تعريف البيانات: create table, drop table, alter table, create index
- لغة التحكم بالبيانات: grant, revoke



تعمل SQL بمبدأ توجيه طلب إلى محرك قاعدة البيانات والحصول على جواب من محرك قاعدة البيانات الذي يُرجع مجموعة نتائج.
توفر SQL مجموعة من التعليمات بحيث يمكن تقسيمها إلى ثلاث لغات فرعية:

لغة معالجة البيانات التي تتضمن التعليمات الخاصة باستعادة البيانات وتعديلها مثل:

Select: وهي مخصصة لقراءة البيانات واستخلاصها من قاعدة البيانات.

Insert: وهي مخصصة لإضافة سجلات جديدة إلى قاعدة البيانات.

Delete: وهي مخصصة لحذف سجل أو مجموعة سجلات من قاعدة البيانات.

Update: وهي مخصصة لتعديل سجل أو مجموعة من السجلات في قاعدة البيانات.

لغة تعريف البيانات المخصصة لتعريف بنية البيانات، وتتضمن تعليمات مثل:

Create table: وهي مسؤولة عن توليد جدول

drop table: وهي مسؤولة عن حذف جدول

alter table: وهي مسؤولة عن تعديل جدول

create index: وهي مسؤولة عن توليد مؤشرات

لغة التحكم بالبيانات التي تُستخدم للتحكم وضبط السماحيات على قاعدة البيانات مثل:

grant
revoke

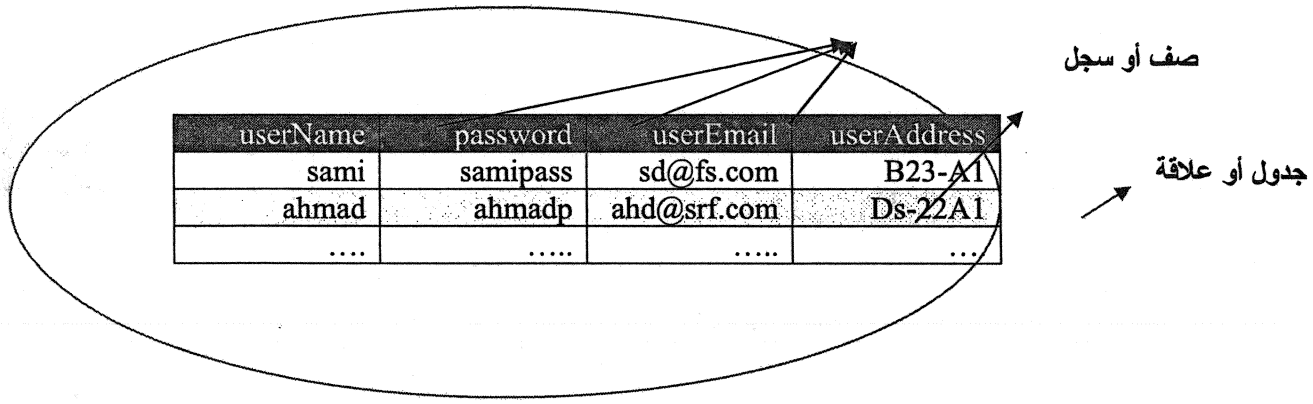
تمتلك قواعد البيانات العلائقية، حسب ما ورد في نموذج كودد، مايلي:

- مجموعة من بنى المعطيات المُستَخدَمة في قاعدة البيانات. أهمها:
  - العلاقة ويتم تمثيلها بالجدول
  - البيانات ويتم تمثيلها بالمعلومات المحتواة ضمن حقول الجدول
  - الخواص ويتم تمثيلها بالأعمدة أو حقول الجدول
  - الصفوف ويتم تمثيلها بالسجلات في الجداول.
  - النطاق ويتم تمثيلها بالمجال الذي يمكن لقيم الخواص أن تتحرك ضمنه.

- توابع تساعد على الوصول إلى البيانات وتساعد على تعديلها واسترجاعها

- مجموعة من القواعد التي تنظم العمليات التي يمكن إجرائها على قاعدة البيانات

مثال: جدول المستخدمين Users :



تعليمة Select 1

تُعتبر تعليمة Select من أشهر تعليمات اللغة وأكثرها استخداماً. تُستخدَم هذه التعليمة لاستعادة وانتقاء مجموعة من البيانات من قاعدة البيانات وذلك بإعادة جدول يحتوي مجموعة البيانات المطلوبة

تأخذ تعليمة Select الصيغة:

```
Select [ field1,field2,..] from [table_name];
```

- تُستخدم إشارة * كبديل لأسماء الحقول (عادة لاتنصح باستخدامها في الحالات التطبيقية لأنها تُحمّل برنامج إدارة قاعدة البيانات عبء تحديد الحقول وتحديد عددها وأسماءها).
- يُستخدم تعبير Distinct لاستعادة جميع السجلات مع إلغاء التكرار في السجلات المعادة.
- يُستخدم التعبير Order by لترتيب السجلات المُعادة ترتيباً تصاعدياً أو تنازلياً حسب التعبير المرافق المستخدم: ASC للترتيب التصاعدي أو DESC للترتيب التنازلي.
- في حال الرغبة باستخدام أسماء بديلة لحقول جدول القيم المعادة نستخدم التعبير AS.

مثال:

إذا كان لدينا قاعدة بيانات تحتوي جدول يدعى Users وأردنا استعادة بيانات حقلي username و password من جميع سجلات هذا الجدول، تكون عبارة SQL كما يلي:

```
Select username, password from Users
```

لاستعادة جميع السجلات من الجدول:

```
Select * from Users
```

لاستعادة جميع بيانات الحقل UserName مع إلغاء التكرار في السجلات المُعادة:

```
Select Distinct UserName from Users
```

لاستعادة مجموعة من السجلات مرتبة ترتيباً تصاعدياً وفق أحد الحقول:

```
Select userName, Password from users order by userName  ASC
```

لاستخدام اسم بديل للحقل userName في جدول القيم المعادة هو Names نكتب التعبير:

```
Select username As Names from users
```

تُعتبر تعليمة Select من أشهر تعليمات اللغة وأكثرها استخداماً. تُستخدَم هذه التعليمة لاستعادة وانتقاء مجموعة من البيانات من قاعدة البيانات وذلك بإعادة جدول يحتوي مجموعة البيانات المطلوبة

- تُستخدم إشارة * (star) كبديل لأسماء الحقول (عادة لاننصح باستخدامها في الحالات التطبيقية لأنها تُحمّل برنامج إدارة قاعدة البيانات عبء تحديد الحقول وتحديد عددها وأسماءها).
- يُستخدم تعبير Distinct لاستعادة جميع السجلات مع إلغاء التكرار في السجلات المعادة.
- يُستخدم التعبير Order by لترتيب السجلات المُعادة ترتيباً تصاعدياً أو تنازلياً حسب التعبير المرافق المستخدم: ASC للترتيب التصاعدي أو DESC للترتيب التنازلي.
- في حال الرغبة باستخدام أسماء بديلة لحقول جدول القيم المُعادة نستخدم التعبير AS.

تعليمة SELECT 2

الكلمة المفتاحية WHERE:

نستخدم الكلمة المفتاحية Where مع تعليمة Select لاستعادة مجموعة من السجلات التي تحقق شرط أو مجموعة من الشروط التي نعبر عنها بعبارة شرطية.

تستخدم الكلمة المفتاحية where الصيغة:

```
Select * from users where condition;
```

- تُعيد العبارة الشرطية قيمة منطقية (صح أو خطأ)

- يمكن للعبارة الشرطية أن تتضمن عمليات مقارنة مثل (=< , =< , < , > , < > , =) ويتم ضم السجل الذي يحققها إلى جدول النتائج.

الكلمات المفتاحية Like وbetween

- تُستخدم الكلمة المفتاحية Like ضمن العبارة الشرطية، كشرط لوجود مثيل. غالباً ما تُستخدَم هذه الكلمة مع إشارة (%)، التي تضاف إلى القيمة التي نبحث عن مثيلاتها، كبديل عن أي رقم من الأرقام أوالأحرف.

- تُستخدم الكلمة المفتاحية Between ضمن العبارة الشرطية، كشرط لوجود قيمة محصورة بين قيمتين محددتين

- تقبل الكلمة المفتاحية where أكثر من شرط يفصل بينها عمليات منطقية مثل AND أو OR ويمكن أن يسبق الشرط العملية NOT لنفيه.

مثال:

إذا أردنا الحصول على قائمة جميع السجلات التي تحتوي على السلسلة 'am' (بمطابقة جزئية أو بمطابقة كاملة) في حقل اسم المستخدم يمكن استخدام تعليمة Select التالية:

```
Select * from users where userName like '%am%';
```

للحصول على قائمة جميع السجلات التي تنحصر فيها قيمة حقل العمر بين 15 و25 يمكن كتابة تعليمة Select التالية:

```
Select * from users where userAge between 15 and 25;
```

إذا أردنا الحصول على قائمة جميع السجلات التي تحتوي على السلسلة 'am' (بمطابقة جزئية أو بمطابقة كاملة) في حقل اسم المستخدم، والتي تنحصر فيها قيمة حقل العمر بين 15 و25 يصبح المثال كما يلي:

```
Select * from users where userName like '%am%'
 And
userAge between 15 and 25;
```

ملاحظة: في Access نستخدم * بدل %

الكلمة المفتاحية WHERE:

نستخدم الكلمة المفتاحية Where مع تعليمة Select لاستعادة مجموعة من السجلات التي تحقق شرط أو مجموعة من الشروط التي نعبر عنها بعبارة شرطية.

- تُعيد العبارة الشرطية قيمة منطقية (صح أو خطأ)

- يمكن للعبارة الشرطية أن تتضمن عمليات مقارنة مثل (=, <>, >, <, <=, >=) ويتم ضم السجل الذي يحققها إلى جدول النتائج.

الكلمات المفتاحية Like وbetween

- تُستخدم الكلمة المفتاحية Like ضمن العبارة الشرطية، كشرط لوجود مثيل. غالباً ما تُستخدَم هذه الكلمة مع إشارة (%)، التي تضاف إلى القيمة التي نبحث عن مثيلاتها، كبديل عن أي رقم من الأرقام أوالأحرف.
- تُستخدم الكلمة المفتاحية Between ضمن العبارة الشرطية، كشرط لوجود قيمة محصورة بين قيمتين محددتين
- تقبل الكلمة المفتاحية where أكثر من شرط يفصل بينها عمليات منطقية مثل AND أو OR ويمكن أن يسبق الشرط العملية NOT لنفيه.

تعليمة DELETE

تقوم تعليمة Delete بحذف سجل أو مجموعة من السجلات من جدول ما.

تأخذ تعليمة Deleteالصيغة:

```
Delete from [table_name]
```

مثال:

للحذف من جدول users نستخدم تعليمة Delete التالية:

```
Delete from Users
```

تُعتبر التعليمة الواردة في المثال السابق خطرة لأنها ستقوم بحذف جميع السجلات من الجدول Users. لذا نحتاج إلى الكلمة المفتاحية Where لتحديد شرط حذف هذه السجلات.

فإذا كنا نريد حذف السجل الخاص بالمستخدم ذي اسم الدخول 'Ahmed' يصبح مثالنا كالتالي:

```
Delete from Users where username='Ahmed';
```

التعليمة Delete

تقوم تعليمة Delete بحذف سجل أو مجموعة من السجلات من جدول ما.

تعليمة INSERT

تُستخدم تعليمة Insert لإدراج سجل في جدول محدد.

تأخذ تعليمة Insert الصيغة:

```
insert into table_name values ( value1,value2,value3,…);
```

حيث تُمثل value1,..value n القيم التي سوف يتم تخزينها في السجل تبعاً لترتيب حقوله.

كما يمكننا استخدام صيغة بديلة تُمكننا من تحديد حقول السجل التي نود إدراج البيانات فيها فقط:

```
Insert into table_name (field1,field2…)
 values (value1,value2,..);
```

يمكن لتعليمة Insert إدراج أكثر من سجل بأمر واحد ولكن ستحتاج إلى استخدام ما ندعوه الاستعلامات الفرعية (Sub queries) التي سنأتي على ذكرها لاحقاً.

مثال:

لإدراج سجل كامل في الجدول users نستخدم الصيغة:

```
insert into Users values
('adel','adelPassword',33,'adel@yahoo.com');
```

أما الحل البديل الذي يُمكننا من تحديد حقول السجل التي نود إدراج البيانات ضمنها وترتيبها فهو:

```
insert into Users (username,password)
values ('adel','adelPassword');
```

لإدراج جميع النتائج، التي أعادها الاستعلام عن جميع حقول الجدول otherUserTable، ضمن الجدول users نستخدم العبارة:

```
Insert into users select * from OtherUserTable
```

تُستخدم تعليمة Insert لإدراج سجل في جدول محدد

يمكن لتعليمة Insert إدراج أكثر من سجل بأمر واحد ولكن ستحتاج إلى استخدام ما ندعوه الاستعلامات الفرعية (Sub queries) التي سنأتي على ذكرها لاحقاً.

تعليمة Update

تُستخدم تعليمة Update من تعديل البيانات في سجل أو في مجموعة من السجلات.

تأخذ تعليمة Update الصيغة:

```
Update table_name Set
Field1= new_field_value1 ,
Field2= new_field_value 2;
```

يمكن استخدام الكلمة المفتاحية where مع تعليمة Update لتصبح الصيغة:

```
Update table_name Set
Field1= new_field_value1 ,
Field2= new_field_value 2
Where condition ;
```

مثال:

```
Update Users set username='sami' , password='sami pass'
```

سيقوم مثالنا بتعديل قيمة اسم المستخدم وكلمة العبور في كل السجلات وفقاً للقيم المعطاة، لذلك لابد هنا من الانتباه إلى استخدام التعبير where ليعطي شرط التعديل كما يلي:

```
Update Users set password='sami pass' where username='sami'
```

تُستخدم تعليمة Update من تعديل البيانات في سجل أو في مجموعة من السجلات. ويمكن استخدام الكلمة المفتاحية where مع تعليمة Update لتحديد شروط التعديل.

بعض الملاحظات العملية

- تتطلب بعض برامج إدارة قواعد البيانات إنهاء كل تعليمة SQL بـ . (;) بينما يضيفها البعض الآخر تلقائياً.
- من المهم استخدام تعليقات SQL وخصوصاً عند كتابة نصوص SQL تحتوي على عدد كبير من الأسطر والتعليمات. يمكن إضافة التعليق كما يلي:

```
Select * from users;  -- this is the comment
```

- من المهم تلافي استخدام أسماء أعمدة (حقول) حاوية على فراغات. أما في الحالات الاضطرارية، فيمكن استخدام إشارات تنصيص أو أقواس مربعة لإحاطة اسم الحقل (أقواس في oracle وإشارة تنصيص في Acess وSQL server) مثال:

```
Select [user name] from users ;
```

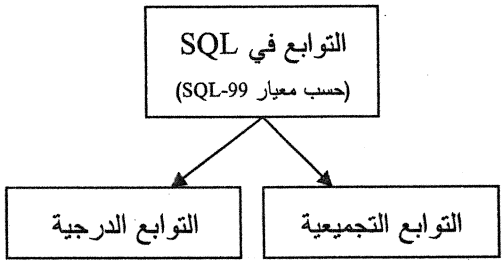
تتطلب بعض برامج إدارة قواعد البيانات إنهاء كل تعليمة SQL بـ . (;) بينما يضيفها البعض الآخر تلقائياً.
ومن المهم استخدام تعليقات SQL وخصوصاً عند كتابة نصوص SQL تحتوي على عدد كبير من الأسطر والتعليمات.
كذلك، من المهم تلافي استخدام أسماء أعمدة (حقول) حاوية على فراغات. أما في الحالات الاضطرارية، فيمكن استخدام إشارات تنصيص أو أقواس مربعة لإحاطة اسم الحقل (أقواس في oracle وإشارة تنصيص في Acess وSQL server).

التابع هو عبارة عن تعبير رياضي ياخذ مجموعه من قيم الدخل التي ندعوها معاملات، ويعيد قيمه خرج وحيدة ندعوها قيمه التابع. تتعلق قيمة التابع (أي الخرج) بمعاملاته (أي بالدخل)، كحال التابع الذي يقوم بحساب مجموع قيم عددية.

التوابع في SQL

تُقسَم توابع SQL، حسب معيار SQL-99، إلى نوعين:

- التوابع التجميعية: مثال: $f(x,y,z)=x+y+z$
- التوابع الدرجية: مثال: $f(x) = |x|$



التابع هو عبارة عن تعبير رياضي يأخذ مجموعة من قيم الدخل التي ندعوها مُعاملات، ويعيد قيمة خرج وحيدة ندعوها قيمة التابع. تتعلق قيمة التابع (أي الخرج) بمُعاملاته (أي بالدخل)، كحال التابع الذي يقوم بحساب مجموع قيم عددية.

التوابع في SQL

تُقسَم توابع SQL حسب معيار SQL-99 إلى نوعين:

- التوابع التجميعية: وهي التوابع التي تأخذ كمُعاملات مجموعة من القيم وتعيد قيمة وحيدة، مثل التابع الذي يحسب مجموع أعداد حقيقية.
- التوابع الدرجية وهي التوابع التي تأخذ مُعاملاً وحيداً وتُعيد قيمة وحيدة، مثل تابع القيمة المُطلقة لعدد حقيقي.

توابع SQL التجميعية

من أهم توابع SQL التجميعية التوابع التالية:

التابع	استخدامه
AVG(expression)	يقوم بحساب معدل القيم لحقل معين
COUNT(expression)	يقوم بحساب عدد البيانات الخاصة بحقل معين
MIN(expression)	يقوم بإعادة القيمة الصغرى من قيم حقل معين
MAX(expression)	يقوم بإعادة القيمة العظمى من قيم حقل معين
SUM(expression)	يقوم بحساب مجموع قيم حقل معين

من أهم توابع SQL التجميعية التوابع التالية:

التابع AVERAGE الذي يُستخدم لحساب المتوسط الحسابي لقيم حقل معين.

التابع COUNT الذي يُستخدم لحساب عدد البيانات الخاصة بحقل مُعين.

التابع MIN الذي يُستخدم لحساب القيمة الصغرى من قيم حقل معين.

التابع MAX الذي يُستخدم لحساب القيمة العظمى من قيم حقل معين.

التابع SUM الذي يإستخدم لحساب مجموع قيم حقل معين.

سنستعرض صيغة كل تابع من التوابع السابقة وأسلوب عمله بالتفصيل في الفقرات اللاحقة.

التابع AVG

يحسب تابع AVG المتوسط الحسابي لقيم حقل معين وذلك بتقسيم مجموع قيم هذا الحقل على عددها.

صيغة التابع:

```
select avg([ALL | DISTINCT]column_name) from table_name
```

- يُستخدم الخيار All للحصول على المتوسط الحسابي لجميع القيم بما فيها القيم المكررة. يُعتبر هذا الخيار هو الخيار التلقائي في حال عدم تحديد أي من الخيارين Distinct أو All.

- يُستخدم الخيار Distinct للحصول على المتوسط الحسابي لجميع القيم، مع استبعاد أي تكرار لقيمة ما.

مثال:

إذا كان لدينا الجدول علامات الطلاب grades الذي يحتوي الحقول studentName وstudentGrade وstudentClass.

للحصول على المتوسط الحسابي لجميع الطلاب، نستخدم التعبير:

```
select avg(studentGrade) from grades
```

إذا أردنا الحصول على المتوسط الحسابي لعلامات الطالب "عادل" مع استبعاد أي تكرار لعلامة من علاماته، يصبح التعبير:

```
select avg(distinct studentGrade) form grades where studentName =
'adel"
```

انتبه:

- قد لا تعمل خيارات All وDistinct على قواعد بيانات MS Access.
- قد تختلف القيمة التي يعيدها تابع AVG من نظام إدارة قواعد بيانات إلى آخر بسبب تحويل النتيجة في الغالب إلى نفس نوع الحقل الذي تم تطبيق التابع عليه.

يحسب تابع **AVG** المتوسط الحسابي لقيم حقل معين وذلك بتقسيم مجموع قيم هذا الحقل على عددها.

- يُستخدم الخيار All للحصول على المتوسط الحسابي لجميع القيم بما فيها القيم المكررة. يُعتبر هذا الخيار هو الخيار التلقائي في حال عدم تحديد أي من الخيارين Distinct أو All.

- يُستخدم الخيار Distinct للحصول على المتوسط الحسابي لجميع القيم، مع استبعاد أي تكرار لقيمة ما.

التابع COUNT

يحسب التابع COUNT عدد البيانات الموجودة في الجدول من أجل حقل معين.

صيغة التابع:

```
select count([* | ALL | DISTINCT]column_name) from table_name
```

- يُستخدم الخيار All عندما نريد الحصول على عدد البيانات الموجودة في الجدول، بالنسبة لحقل معين، مع استبعاد القيم التي تساوي Null. يُعتبر هذا الخيار هو الخيار التلقائي في حال عدم تحديد أي من الخيارين Distinct أو All.

- يُستخدم الخيار Distinct عندما نريد الحصول على عدد البيانات الموجودة في الجدول، بالنسبة لحقل معين، مع استبعاد القيم التي تساوي Null واستبعاد أي تكرار في قيمة ما.

- يُستخدم الخيار * عندما نريد الحصول على عدد البيانات الموجودة في الجدول بالنسبة لحقل معين بما فيها البيانات ذات القيمة Null.

مثال:

إذا كان لدينا الجدول علامات الطلاب grades الذي يحتوي الحقول studentName وstudentGrade وstudentClass. وإذا أردنا الحصول على عدد الطلاب نستخدم التعليمة:

```
select count(*) from grades
```

لكن المثال السابق سيحسب عدد الطلاب اعتماداً على السجلات التي قد تحتوي على القيم Null. لذلك إذا أردنا الحصول على عدد الطلاب توجب علينا حساب عدد البيانات الموجودة في الجدول بالنسبة لحقل اسم الطالب شرط ألا يكون اسم الطالب مساوياً لـ Null (مما يعني أن الطالب موجود فعلاً)، تصبح التعليمة عندها:

```
select count(all studentName) from grades
```

لكن، قد يكون أحد أسماء الطلاب مكرراً، لذا نستخدم التعليمة التالية لنحصل على عدد الطلاب الحقيقي:

```
select count(distinct studentName) from grades
```

انتبه:

قد لا تعمل خيارات All وDistinct على قواعد بيانات MS Access.

يحسب التابع COUNT عدد البيانات الموجودة في الجدول من أجل حقل معين.

- يُستخدم الخيار All عندما نريد الحصول على عدد البيانات الموجودة في الجدول، بالنسبة لحقل معين، مع استبعاد القيم التي تساوي Null. يُعتبر هذا الخيار هو الخيار التلقائي في حال عدم تحديد أي من الخيارين Distinct أو All.

- يُستخدم الخيار Distinct عندما نريد الحصول على عدد البيانات الموجودة في الجدول، بالنسبة لحقل معين، مع استبعاد القيم التي تساوي Null واستبعاد القيم المكررة.

- يُستخدم الخيار * عندما نريد الحصول على عدد البيانات الموجودة في الجدول، بالنسبة لحقل معين، بما فيها البيانات ذات القيمة Null.

التابع MIN والتابع MAX

يُعيد التابع MIN القيمة الصغرى من قيم حقل معين.

صيغة التابع:

```
select min(column_name) from table_name
```

يُعيد التابع MAX القيمة العظمى من قيم حقل معين.

صيغة التابع:

```
select max(column_name) from table_name
```

لا تأثير للخيارين All وDistinct على التوابع MIN وMAX رغم أنه بالإمكان استخدامهما. فالقيمة العظمى أو القيمة الصغرى، لقيم حقل، تبقى نفسها حتى ولو كان هناك تكرار في قيم الحقل وحتى ولو كان هناك قيم غير محددة (أي تساوي Null).

مثال:

إذا كان لدينا الجدول علامات الطلاب grades الذي يحتوي الحقول studentName وstudentGrade وstudentClass، وإذا أردنا الحصول على أخفض علامة نستخدم التعليمة:

```
select min(studentGrade) from grades
```

و إذا أردنا الحصول على أعلى علامة نستخدم التعليمة:

```
select max(studentGrade) from grades
```

يُعيد التابع MIN القيمة الصغرى من قيم حقل معين في حين يُعيد التابع MAX القيمة العظمى من قيم حقل معين.

تجدر الإشارة إلى عدم وجود أي تأثير للخيارين All وDistinct على التوابع MIN وMAX رغم أنه بالإمكان استخدامهما. فالقيمة العظمى أو القيمة الصغرى لقيم حقل، تبقى نفسها، حتى ولو كان هناك تكرار في قيم الحقل وحتى ولو كان هناك قيم غير محددة (أي تساوي Null).

التابع SUM

يحسب التابع SUM مجموع قيم حقل معين.

صيغة التابع:

```
select sum([ALL | Distinct]column_name) from table_name
```

- يُستخدم الخيار All عندما نريد الحصول على مجموع قيم حقل معين بما فيها القيم المكررة. يُعتبر هذا الخيار هو الخيار التلقائي في حال عدم تحديد أي من الخيارين Distinct أو All.
- يستخدم الخيار Distinct عندما نريد الحصول على مجموع قيم حقل معين مع استبعاد أي تكرار في القيم.

مثال:

إذا كان لدينا الجدول علامات الطلاب grades الذي يحتوي الحقول studentName وstudentGrade وstudentClass، وإذا أردنا الحصول على مجموع علامات الطلاب نستخدم التعليمة:

```
select sum(studentGrade) from grades
```

انتبه:

لا يمكن استخدام التابع SUM على أنواع حقول ذات أنماط غير مناسبة كونه يعتمد عملية الجمع الحسابي، أي لا يمكننا، على سبيل المثال، استخدام تابع SUM مع حقل من نمط سلسلة محارف.

يحسب التابع SUM مجموع قيم حقل معين:

- يُستخدم الخيار All عندما نريد الحصول على مجموع قيم حقل معين بما فيها القيم المكررة. يُعتبر هذا الخيار هو الخيار التلقائي في حال عدم تحديد أي من الخيارين Distinct أو All.

- يستخدم الخيار Distinct عندما نريد الحصول على مجموع القيم حقل معين مع استبعاد أي تكرار في القيم.

تجميع البيانات 1

عندما نتكلم عن التوابع التجميعية فلابد لنا أن نتساءل: هل نستطيع أن نطبق التوابع التجميعية على مجموعات جزئية من السجلات بدلاً من تطبيقها على كامل السجلات؟

فإذا كان لدينا جدول منتجات (Products) وأردنا حساب مجموع كلف المنتجات (unitCost) التي نحصل عليها من المورد الأول، وحساب مجموع كلف المنتجات التي نحصل عليها من المورد الثاني، وحساب مجموع كلف المنتجات التي نحصل عليها من المورد الثالث، فإننا سنحتاج لكتابة ثلاث تعليمات منفصلة تعتمد على التابع التجميعي SUM كما يلي:

```
Select sum(unitCost) from products where supplier ID= 1;

Select sum(unitCost) from products where supplier ID= 2;

Select sum(unitCost) from products where supplier ID= 3;
```

لكن هل يمكننا أن نصل إلى تركيب شبيه بالتركيب الظاهر في المثال السابق باستخدام تعليمة واحدة؟

تغطي SQL هذا المتطلب بآلية تساعد على تقسيم السجلات إلى مجموعات جزئية وتساعد على تطبيق التوابع التجميعية على كل مجموعة جزئية على حدى.

عندما نتكلم عن التوابع التجميعية فلابد لنا أن نتساءل: هل نستطيع أن نطبق التوابع التجميعية على مجموعات جزئية من السجلات بدلاً من تطبيقها على كامل السجلات؟
فإذا كان لدينا جدول منتجات، وأردنا حساب مجموع كلف المنتجات التي نحصل عليها من الموزد الأول، وحساب مجموع كلف المنتجات التي نحصل عليها من المورد الثاني، وحساب مجموع كلف المنتجات التي نحصل عليها من المورد الثالث، فإننا سنحتاج لكتابة ثلاث تعليمات منفصلة تعتمد على التابع التجميعي SUM، لكن هل يمكننا أن نصل إلى نفس النتيجة بتعليمة واحدة فقط؟

تغطي SQL هذا المتطلب بآلية تساعد على تقسيم السجلات إلى مجموعات جزئية وتساعد على تطبيق التوابع التجميعية على كل مجموعة جزئية على حدى.

تجميع البيانات 2

لتجميع البيانات في SQL نستخدم تعليمة Group by والتي تأخذ الصيغة التالية:

```
Select columnA, aggFunc (aggFuncSpec) from table
where whereSpec
Group by columnA
```

مثال:

إذا كان لدينا جدول Sales يحتوي أسماء المنتجات ProductName وكمية البيع في مراكز مختلفة Quantity، يمكننا كتابة التعليمة التي تعطي الكمية المباعة من كل منتج في جميع المراكز بعد تاريخ معين.

```
Select productName, sum (quantity) from sales
where saleDate > #20/5/2013 #
Group by productName
```

لتجميع البيانات في SQL نستخدم تعليمة Group by.

الكلمة المفتاحية Having

- عندما نفكر في وضع شرط ما على استعلام معين، يرد إلى ذهننا استخدام الكلمة المفتاحية where مع شرط مناسب.

```
Select field_name from table_name where condition
```

ولكن هذا الحل البديهي يكون قاصراً في بعض الحالات مثل الحالة التي يكون فيها أحد عناصر الشرط تابعاً تجميعياً.

- نستخدم الكلمة المفتاحية Having في حال أردنا أن نضع شرطاً على استعلام ما، بحيث يكون أحد عناصر الشرط تابعاً تجميعياً.

- نستعمل Having ضمن الصيغة التالية:

```
Select columnA, aggFunc (aggFuncSpec) from table
where whereSpec
Group by columnA
Having filterCondition
```

انتبه: لا تلغي الكلمة المفتاحية Having دور where بل يمكن استخدامهما معاً في صيغة واحدة كما يظهر في الصيغة السابقة

مثال:

إذا كان لدينا الجدول StudentsGrade الحاوي على أرقام الطلاب (StudentNumber) وعلى علاماتهم (grade) في مختلف المواد، وأردنا معرفة أي من الطلاب قد حصل على معدل جيد جداً (أكبر من 70) أو سيئ (أصغر من 50)، سيخطر لنا استخدام التعبير التالي:

```
Select studentNumber, Avg(grade) as averageMark from studentGrades
Where avg(grade)>70 or avg(grade)<50
Group by studentNumber
```

إن الحل السابق خاطئ، لأن where تقوم بعملية الترشيح قبل تنفيذ التابع التجميعي (avg) وليس بعده، لذا نحن بحاجة لاستخدام Having لحلّ هذه المسألة. يكون الحل كما يلي:

```
Select studentNumber, Avg(grade) as averageMark from studentGrades
Group by studentNumber
Having avg(grade)>70 or avg(grade)<50
```

عندما نفكر في وضع شرط ما على استعلام معين، يرد إلى ذهننا استخدام الكلمة المفتاحية where مع شرط مناسب.

ولكن هذا الحل البديهي يكون قاصراً في بعض الحالات مثل الحالة التي يكون فيها أحد عناصر الشرط تابعاً تجميعياً

عندها يأتي دور الكلمة المفتاحية having في حال أردنا أن نضع شرطاً على استعلام ما، بحيث يكون أحد عناصرالشرط تابعاً تجميعياً.

عموماً، لا بد من الإشارة إلى أن الكلمة المفتاحية Having لا تلغي دور الكلمة المفتاحية where بل يمكن استخدامهما معاً في صيغة واحدة.

التعبير Top N

يُستخدم التعبير (Top N) كمرافق للتوابع التجميعية ولكن استخدامه لا يقتصر عليها فقط.
يُعيد هذا التعبير أول N سجل من نتيجة الاستعلام.
يأخذ هذا التعبير الصيغة:

```
Select top N field1, field2 … from table_name
```

مثال:

لنفرض أن لدينا جدول StudentsGrade التالي الذي يحتوي على أسماء الطلاب (StudentName) وعلاماتهم (StudentMark) في جميع المواد، ولنفرض أننا أردنا معرفة أسماء الطلاب الخمسة الأوائل مُرتبة، بحسب معدلاتهم، ترتيباً تنازلياً (معدل الطالب هوالمتوسط الحسابي لجميع علاماته).

studentName	studentMark	subject
ahmad	15	math
adel	22	math
ahmad	26	history
...	...	...

نستخدم الصيغة:

```
Select top 5 studentName, avg(studentMark)
From studentsGrades
Group by studentName
order by avg(studentMark) DESC
```

يُستخدم التعبير (Top N) كمرافق للتوابع التجميعية ولكن استخدامه لا يقتصر عليها فقط. ويُعيد هذا التعبير أول N سجل من نتيجة الاستعلام.

استخدام التعبير Top N في مختلف أنظمة إدارة قواعد المعطيات

تختلف الصيغة الخاصة بالتعبير (Top N)، من نظام إدارة قواعد بيانات إلى آخر.

- ففي Mysql تكون الصيغة على الشكل التالي:

```
Select field1, field2 from table_name
Limit 0,N
```

حيث تشير الكلمة المفتاحية limit إلى مجال أرقام السجلات التي نريد الحصول عليها (سجل البداية 0 وسجل النهاية N) من مجموعة السجلات التي تعيدها تعليمة Select.

- أما في قواعد بيانات DB2 فتكون الصيغة على الشكل التالي:

```
Select field1, field2 from table_name
Fetch first N rows only
```

حيث يحدد المعامل N في التعبير "fetch first N rows only" عدد السجلات التي نريد الحصول عليها وذلك ابتداءاً من السجل الأول.

- وتعتمد الصيغة في قواعد بيانات Oracle على الكلمة المفتاحية rowNum وهي عبارة عن قيمة تلقائية يولدها نظام Oracle وتحدد ترتيب السجلات التي نريد الحصول عليها ضمن مجموعة السجلات التي تعيدها تعليمة Select. فتكون الصيغة على الشكل:

```
Select field1, field2 where rowNum<= N
```

مثال:

في حال أردنا الحصول على الأرقام الثلاثة الأولى التي سجلت أكبر عدد من الاتصالات لدى شركة هواتف محمولة وذلك اعتباراً من سجل اتصالات الزبائن (callLog) المؤلف من حقول: إسم المستخدم (userName)، ورقم هاتف المستخدم (phoneNumber). وبفرض أن نظام إدارة قواعد المعطيات المُستخدم هو نظام Oracle، نستخدم الصيغة:

```
Select phoneNumber, count(userName) from callLog
Group by phoneNumber
Order by count(userName)
Where rowNum<= 3
```

تختلف الصيغة الخاصة بالتعبير (Top N)، من نظام إدارة قواعد بيانات إلى آخر.

ففي Mysql نستخدم الكلمة المفتاحية limit التي تشير إلى مجال أرقام السجلات التي نريد الحصول عليهاK من مجموعة السجلات التي تعيدها تعليمة Select.

أما في قواعد بيانات DB2 فنستخدم التعبير "fetch first N rows only" الذي يشير إلى عدد السجلات التي نريد الحصول عليها وذلك ابتداءاً من السجل الأول.

وتعتمد الصيغة في قواعد بيانات Oracle على الكلمة المفتاحية rowNum وهي عبارة عن قيمة تلقائية يولدها نظام Oracle وتحدد ترتيب السجلات التي نريد الحصول عليها ضمن مجموعة السجلات التي تعيدها تعليمة Select.

مسألة:

تحتفظ شركة متعددة الفروع بسجل خاص بالمهام والأعمال التي نفذها موظفو الشركة والدخل الناتج عن كل مهمة من هذه المهام.
بفرض أن جدول المعلومات المتوفرة هو جدول المهام tasks التالي:

taskID	taskDate	taskIncom	taskHandler
1	27/1/2003	10000	adel
...	...	...	...

حيث يعبر الحقل taskID عن رقم المهمة
ويعبر الحقل taskDate عن تاريخ المهمة
ويعبر الحقل taskIncom عن الدخل الناتج عن المهمة
ويعبر الحقل taskHandler عن اسم الموظف الذي نفذ المهمة

والمطلوب الحصول على جدول يحدد:

- عدد الأعمال التي قام بها كل موظف.
- الربح الكلي الذي حصلت عليه الشركة من عمل هذا الموظف وذلك خلال العام 2004.

مع ترتيب الجدول على نحو يظهر فيه الموظف الذي نفذ المهمة التي حققت أعلى دخل للمؤسسة، في أول سجل في الجدول.

الحل:

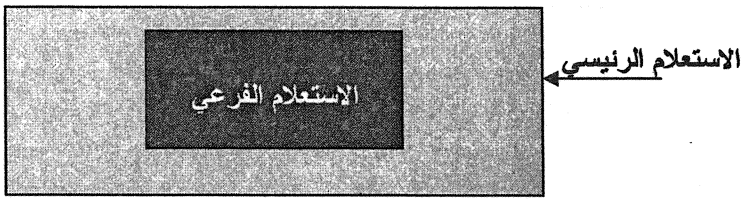
سيظهر في جدول النتيجة معلومات حول اسم الموظف، وعدد المهمات التي أوكلت له، ومجموع الدخول التي حصلت عليها الشركة من عمل الموظف، وقيمة الدخل الأعظمي الذي حصل عليه الموظف من مهامه، مع ترتيب قيم الدخول الأعظمية ترتيباً تنازلياً، فيكون أول اسم في الجدول هو اسم الموظف الذي نفذ المهمة التي حققت أعلى دخل للمؤسسة.

يمكن الحصول على الحل المطلوب بالصيغة التالية:

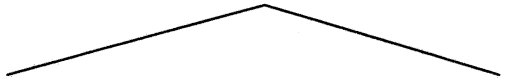
```
Select taskHandler, count(taskID), sum(taskIncom), max(taskIncom)
From tasks
Group by taskHandler
Where taskDate between#01/01/2004# and#31/12/2004#
Order by max(taskIncom) DESC
```

الاستعلامات الفرعية

الاستعلام الفرعي هو أي استعلام يتم تضمينه في استعلام آخر.



هناك نوعان أساسيان من الاستعلامات الفرعية



انتبه:

- يمكن للاستعلام الفرعي أن يحتوي استعلاماً فرعياً آخر.
- ليس من الضرورة أن تَستخدم الاستعلامات الفرعية نفس الجداول التي تستخدمها الاستعلامات الرئيسية.
- لا تدعم قواعد بيانات MySQL الاستعلامات الفرعية، بل تستعيض عنها بـ . Join التي سنأتي على ذكرها لاحقاً.

الاستعلام الفرعي هو أي استعلام يتم تضمينه في استعلام آخر.

هناك نوعان أساسيان من الاستعلامات الفرعية:

1- الاستعلامات الفرعية المرتبطة
2- الاستعلامات الفرعية غير المرتبطة

الاستعلامات الفرعية

1- **الاستعلامات الفرعية المرتبطة باستعلام رئيسي:** هي الاستعلامات الفرعية التي تعتمد في عملها على بيانات من استعلامات رئيسية حاوية لها.

في هذا النوع من الاستعلامات الفرعية، يتم تكرار عملية تنفيذ الاستعلام بعدد مرات مساوٍ لعدد السجلات التي يُعيدها الاستعلام الرئيسي.

مثال:

إذا كان لدينا الجدول Customers الحاوي على معلومات الزبائن، والجدول Orders الحاوي على معلومات الطلبيات. لإظهار قائمة باسم كل زبون (customerName) وعدد الطلبيات لكل زبون نكتب الاستعلام:

```
Select customerName,(select count(*) from Orders
where Orders.customerID=Customers.customerID) from Customers;
```

نلاحظ في المثال السابق بأنه سيتم تكرار تنفيذ الاستعلام الفرعي (الظاهر بين القوسين) بعدد سجلات الزبائن وأن الاستعلام سيعيد عدد الطلبيات لكل زبون.

2- **الاستعلامات الفرعية المستقلة:** وهي الاستعلامات الفرعية التي تكون مستقلة تماماً عن الاستعلامات الرئيسية الحاوية لها، أي أن الاستعلام الفرعي سيُنَفَّذ بشكل كامل ويُمرر القيمة أو مجموعة القيم الناتجة إلى الاستعلام الرئيسي.

مثال:

إذا كان لدينا جدول Students يحتوي أسماء الطلاب (studentName) وأرقامهم التسلسلية (studentID). وجدول آخر Grades يحتوي علامات الطلاب (grade) وأرقامهم التسلسلية (studentID). لإعادة لائحة بأسماء الطلاب الناجحين فقط نكتب الاستعلام:

```
Select studentName from Students where Students.studentID in (select
Grades.studentID from Grades where Grades.grade>=50);
```

(اعتُبرت علامة النجاح 50).

نلاحظ في الاستعلام السابق أنه سيتم تنفيذ الاستعلام الفرعي (بين القوسين) مرة واحدة فقط، وبصورة مستقلة عن عدد السجلات في الجدول Students.

انتبه:

- تحتاج الاستعلامات الفرعية المرتبطة باستعلام رئيسي للكثير من الوقت والمعالجة بالمقارنة مع الاستعلامات غير المرتبطة باستعلامها الرئيسي.
- يتطلب العمل بالاستعلامات الفرعية أحياناً استخدام حقول من أكثر من جدول، فاحرص على عدم استخدام أسماء حقول طموحة بل استخدم الصيغة Table_Name.Field_Name.

1- **الاستعلامات الفرعية المرتبطة باستعلام رئيسي:** هي الاستعلامات الفرعية التي تعتمد في عملها على بيانات من استعلامات رئيسية حاوية لها.
في هذا النوع من الاستعلامات الفرعية، يتم تكرار عملية تنفيذ الاستعلام بعدد مرات مساوٍ لعدد السجلات التي يُعيدها الاستعلام الرئيسي.

2- **الاستعلامات الفرعية المستقلة:** وهي الاستعلامات الفرعية التي تكون مستقلة تماماً عن الاستعلامات الرئيسية الحاوية لها، أي أن الاستعلام الفرعي سيُنَفَّذ بشكل كامل ويُمرر القيمة أو مجموعة القيم الناتجة إلى الاستعلام الرئيسي.

استعمال الاستعلام الفرعي كعمود من أعمدة الاستعلام الرئيسي

يأخذ هذا الاستعلام الصيغة:

```
Select columnA, (subquery) as columnB from Table_Name;
```

في الصيغة السابقة سوف يتم تنفيذ الاستعلام Subquery على كل سجل يعيده الاستعلام الرئيسي. يفيد هذا النوع من الاستعلامات الفرعية في توليد علاقة بين جدول وآخر.
وفقاً لهذه الصيغة يجب ألا يعيد الاستعلام الفرعي أكثر من قيمة واحدة لكل سجل في سجلات الاستعلام الرئيسي.

مثال:
نفرض أن لدينا الجدول Accounts الحاوي على الأرقام التسلسلية للحسابات المصرفية accountID وقيم أرصدة هذه الحسابات accountBalance. ولنفرض أن لدينا الجدول Clients الحاوي على أسماء أصحاب الحسابات clientName، ورقم الحساب لكل زبون accountID. لإظهار قائمة بأرقام الحسابات وأرصدتها وأسماء أصحابها نستخدم الصيغة:

```
Select Accounts.accountID, (select clientName from Clients where
Clients.accountID = Accounts.accountID)as myClientName,
Accounts.accountBalance
from Accounts;
```

يُعتبر المثال السابق عيّنة من الاستعلامات الفرعية المرتبطة باستعلام رئيسي نظراً لكون الاستعلام الفرعي (الظاهر بين الأقواس) لن يعمل وحيداً، بل يحتاج إلى معلومات من الاستعلام الرئيسي وهي قيم Accounts.accountID.

انتبه:

- لاحظ أننا استخدمنا في بعض من الصيغ السابقة أسماء الجداول مع أسماء الحقول مفصولة بنقاط '.' وذلك لمنع حدوث خلط في تحديد تبعية أحد الحقول لأي من الجداول.
- لكي تعمل الصيغة في المثال السابق دون أخطاء، يُشترط أن يعيد الاستعلام الفرعي قيمة واحدة لكل سجل من سجلات الاستعلام الرئيسي.

يُعتبر استعمال الاستعلام الفرعي كعمود من أعمدة الاستعلام الرئيسي، أحد التركيبات المُستخدمة بكثرة في الاستعلامات الفرعية.

استعمال الاستعلامات الفرعية ضمن شرط Where

يأخذ هذا الاستعلام الصيغة:

```
Select columnA, columnB from Table_Name where columnB=(Subquery);
```

في هذه الصيغة أيضاً يجب أن يعيد Subquery قيمة وحيدة.

نلاحظ أن نتيجة الاستعلام الفرعي دخلت كجزء من الشرط في تعبير Where.

مثال:

لدينا الجدول Tickets الحاوي على المخالفات المرورية للعربات والجدول Owners الحاوي على أرقام لوحات العربات carNumber وأسماء أصحابها ownerName. لإظهار اسم صاحب العربة ownerName التي ارتكبت المخالفة رقم 1234، نكتب الصيغة:

```
Select ownerName, Owners.carNumber from Owners
where Owners.carNumber=(select Tickets.carNumber from tickets
where ticketNumber=1234);
```

تدخل نتيجة الاستعلام الفرعي ضمن الشرط في تعبير Where شرط أن يعيد الاستعلام الفرعي قيمة وحيدة.

الاستعلامات الفرعية التي تعيد مجموعة من القيم

كي لا يفشل الاستعلام، اشترطنا في دراستنا للاستعلامات الفرعية حتى الآن، أن يعيد الاستعلام الفرعي قيمة وحيدة.

لكي نتمكن من استخدام الاستعلامات الفرعية التي تعيد أكثر من قيمة نستخدم الصيغة التالية:

```
Select columnA, columnB from Table_Name where columnC IN(Subquery);
```

نلاحظ أننا استخدمنا هنا التعبير IN.

لكن يُشترط في الاستعلام الفرعي من هذا النوع (أي الاستعلام المُمثل بـ . Subquery)، أن يعيد قيم عمود واحد فقط أي يكون من الشكل:

```
Select column1 from Table1;
```

مثال:

لنعد إلى مثال المخالفات المرورية، للحصول على قائمة بأسماء الأشخاص الذين لديهم مخالفات يكون شكل الاستعلام:

```
Select ownerName from Owners where Owners.carNumber IN
(select Distinct Tickets.carNumber from Tickets);
```

استخدمنا هنا الاستعلام الفرعي لإعادة أرقام السيارات التي لها مخالفات في سجل المخالفات مع التخلص من التكرار باستخدام Distinct.

اشترطنا حتى الآن في الاستعلامات الفرعية، سواء تلك المستخدمة كأعمدة في الاستعلام الرئيسي أو تلك المستخدمة ضمن شرط التعبير Where، أن يعيد الاستعلام الفرعي قيمة وحيدة كي لا يفشل الاستعلام.

لكي نتمكن من استخدام الاستعلامات الفرعية التي تعيد أكثر من قيمة نستخدم التعبير IN، لكن يجب أن نراعي أن يعيد الاستعلام الفرعي المستخدم مع التعبير IN حقلاً واحداً فقط (نقصد هنا حقلاً وحداً وليس قيمة واحدة).

استخدام تعبيرات Exists وAll وAny مع الاستعلامات الفرعية

التعبير Exists:

يُستخدم التعبير Exists للتحقق من إعادة الاستعلام الفرعي الذي يليه لأي سجل. ويأخذ التعبير كاملاً القيمة True في حال أرجع الاستعلام الفرعي سجلاً أو أكثر، والقيمة False إذا لم يُرجع الاستعلام الفرعي أي سجل.

يُكتب التعبير Exists وفق الصيغة:

```
Select columnA, columnB from Table_Name where Exists (Subquery);
```

مثال:

إذا كان لدينا جدول Orders بجميع الطلبيات الخاصة بشركة ، والذي يحتوي رقم الطلبية orderID ونوعها orderType ورقم الزبون clientID الذي طلبها. وجدول Clients الحاوي على أسماء الزبائن وأرقامهم clientID. لإظهار قائمة بأسماء الزبائن الذين قاموا سابقاً بإرسال طلبية من نوع "تجهيزات كهربائية"، نستخدم الصيغة:

```
Select Clients.clientName from Clients where Exists
( select * from Orders
where Orders.clientID = Clients.clientID
and
orderType='تجهيزات كهربائية');
```

استخدام تعبيرات Exists وAll وAny مع الاستعلامات الفرعية

التعبير All:

يُستخدم التعبير All للتحقق من كون جميع القيم المعادة من استعلام فرعي، تحقق شرطاً ما في تعبير Where التابع للاستعلام الرئيسي.

يُمكن أن يُكتب التعبير All وفق الصيغة:

```
Select columnA from TableA
where columnA > All from (select columnB from TableB);
```

حيث يتم هنا اختيار السجلات من الجدول A، بحيث تكون كل قيمة من قيم الحقل columnA أكبر من جميع القيم التي يعيدها الاستعلام الفرعي، أي من جميع قيم columnB في الجدول TableB.

مثال:

تريد الجهات المعنية مقارنة الأرقام Time التي سجلها العداؤون في بطولة معينة والمحفوظة مع أسمائهم Name في جدول currentRecords، بالأرقام المسجلة في جدول أرقامهم القديمة oldRecords وذلك لاستخلاص قائمة بأسماء العدائين الذين حطمت أرقامهم جميع الأرقام القديمة oldTime.

للحصول على هذه القائمة، نكتب الصيغة:

```
Select Name from currentRecords
where time < All (select oldTime from oldRecords);
```

استخدام تعبيرات Exists و All وAny مع الاستعلامات الفرعية

التعبير ANY:

يُستخدم التعبير ANY للتحقق من كون قيمة أو أكثر من القيم المعادة من استعلام فرعي، تحقق شرطاً واحداً على الأقل من شروط تعبير Where الخاص بالإستعلام الرئيسي.

يُمكن أن يُكتب التعبير Any وفق الصيغة:

```
Select columnA from TableA
where columnA > ANY from (select columnB from TableB);
```

سيتم هنا اختيار السجلات من الجدول A، حيث تكون كل قيمة من قيم الحقل columnA أكبر من قيمة واحدة على الأقل من القيم المعادة من الاستعلام الفرعي، أي من قيمة واحدة على الأقل من قيم columnB في الجدول TableB.

مثال:

تريد الجهات المعنية مقارنة الأرقام Time التي سجلها العداؤون في بطولة معينة والمحفوظة مع أسمائهم Name في جدول currentRecords، بالأرقام القياسية العالمية المسجلة في جدول bestRecords وذلك لاستخلاص قائمة بأسماء العدائين الذين حطمت أرقامهم أحد الأرقام القياسية bestTime.

للحصول على هذه القائمة نكتب الصيغة:

```
Select Name from currentRecords
where time < ANY (select bestTime from bestRecords);
```

دمج البيانات من استعلامين باستخدام التعابير Union و Intersect وExcept

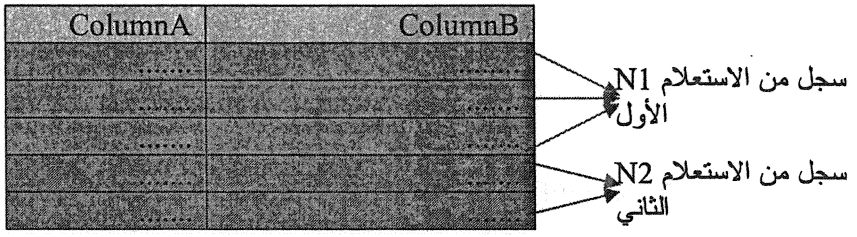
تُستخدم تعابير Union وIntersect و(Except(Minus في دمج استعلامين. وتختلف نتائج هذا الدمج بحسب التعبير المستخدم.

تُستخدم التعابير السابقة وفق الصيغة:

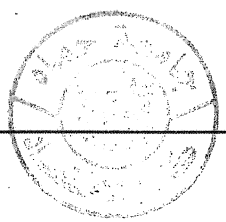
```
select columnA,columnB from tableA
Operator
Select columnC,columnD from tableB;
```

حيث يعبر Operator عن أحد هذه التعابير.

تكون هنا قائمة القيم المعادة على الشكل:



حيث N1 وN2 أكبر أو تساوي الصفر.



الاستعلام عن أكثر من جدول واستخدام الربط الداخلي والخارجي

سبق ورأينا في الجلسات السابقة، أن الاستعلام عن أكثر من جدول باستخدام الاستعلامات الفرعية قد وفر قدرات جيدة على معالجة البيانات. ولكن هذه القدرات قد لا تمكننا دائماً من الحصول على كل النتائج التي نحتاجها. كما أن الصيغة قد تصبح صعبة الفهم بعض الشيء وقد تؤدي إلى انخفاض مستوى الأداء أحياناً.

توفر SQL إمكانية الاستعلام عن جداول متعددة في وقت واحد باستخدام صيغة أبسط ندعوها الربط.

لا تستطيع الصيغة الجديدة استبدال كل التقنيات التي تؤمنها الاستعلامات الفرعية، ولكنها تمثل الحل الأمثل في بعض الحالات، وخاصةً في الاستعلامات التي تربط بين سجلات من جداول مختلفة.

نبدأ فيما يلي بعرض حالات الربط البسيطة لنصل بعد ذلك إلى الحالات الأكثر تعقيداً:

- الربط البسيط
- الربط بالتساوي
- الربط بعدم المساواة
- الربط الخارجي

الربط البسيط

- الربط البسيط هو استعلام عن أكثر من جدول في صيغة واحدة دون استعمال أي شرط.
- إن أبسط صيغة للتعبير عن الربط البسيط هي:

```
Select Table1.Column1, Table2.Column2 from Table1, Table2;
```

إذا فكرنا في استثمار هذه الصيغة لإعادة إسم، وصف كل طالب من الجدولين Names, Classes أول ما سيحضر إلى أذهاننا هو كتابة الصيغة:

```
Select class, Name from Classes, Names;
```

يستخدم هذا الاستعلام الربط البسيط ولكنه للأسف لن يعيد النتيجة التي نحتاجها.

- إن آلية عمل الربط البسيط في حال عدم وجود أي شرط مرافق هي عبارة عن عملية جداء ديكارتي لقيم الحقول المحددة من الجدولين. فإذا كانت قيم الحقل المطلوب من الجدول الأول هي {A, B, C} وقيم الحقل المطلوب من الجدول الثاني هي {D, E, F} فسيكون عدد القيم المعادة 9 قيم هي التالية:
{(A,D), (A,E), (A,F), (B,D), (B,E), (B,F), (C,D), (C,E), (C,F)}

بالنتيجة، إذا كان لدينا 100 سجل في كل من الجدولين المربوطين ربطاً بسيطاً سنحصل على 10000 سجل يعيدها استعلام الربط البسيط.

- يسمى استعلام الربط البسيط أيضاً بالربط المتصالب. ويمكن التعبير عن نفس صيغة الربط السابقة، بالصيغة:

```
Select Table1.Column1, Table2.Column2 from Table1 Cross Join Table2;
```

مثال:

يريد كيميائي اختبار تأثير مجموعة من المواد الكيميائية على مجموعة من المنتجات التي تصنعها الشركة التي يعمل لديها. فإذا كانت أسماء المنتجات (productName) مدرجة في جدول Products وأسماء المواد المراد اختبار تأثيرها (materialName) مدرجة في جدول ChemicalEffects، المطلوب مساعدة هذا الكيميائي في تجهيز لائحة الاختبارات.

الحل:

```
Select productName, materialName from Products, ChemicalEffects;
```

أو

```
Select productName, materialName from Products Cross Join
ChemicalEffects;
```

انتبه:

لا تدعم قواعد بيانات DB2 التعبير Cross Join

الربط البسيط

الربط البسيط هو استعلام عن أكثر من جدول في صيغة واحدة دون استعمال أي شرط.

إن آلية عمل الربط البسيط في حال عدم وجود أي شرط مرافق هي عبارة عن عملية جداء ديكارتي لقيم الحقول المحددة من الجداول المرتبطة. فإذا كانت قيم الحقل المطلوب من الجدول الأول هي {A, B, C} وقيم الحقل المطلوب من الجدول الثاني هي {D, E, F} فسيكون عدد القيم المعادة 9 قيم وهي التالية:
{(A,D), (A,E), (A,F), (B,D), (B,E), (B,F), (C,D), (C,E), (C,F)}

بالنتيجة، إذا كان لدينا 100 سجل في كل من الجدولين المربوطين ربطاً بسيطاً سنحصل على 10000 سجل يعيدها استعلام الربط البسيط.

يسمى استعلام الربط البسيط أيضاً بالربط المتصالب.

الربط بالتساوي

- يُعرّف الربط بالتساوي على أنه الربط البسيط بين سجلات جدول أول، وسجلات جدول ثان اعتماداً على مساواة بين قيمة حقل في سجل من الجدول الأول وقيمة حقل في سجل من الجدول الثاني.

- يُعبّر عن الربط بالتساوي بالصيغة:

```
Select Table1.Column1, Table1.Column2, Table2.Column3
From Table1, Table2 where Table1.Column1 = Table2.Column2;
```

لاحظ أننا قد استخدمنا مساواة بين الحقل الأول من الجدول الأول Table1.Column1 والحقل الثاني من الجدول الثاني Table2.Column2 مما يعني أن عملية الربط لاتستخدم بالضرورة نفس الحقول التي يجب أن يعيدها الاستعلام، أي:
Table1.Column1, Table1.Column2, Table2.Column3

- يمكن الوصول إلى نفس النتيجة السابقة باستخدام الصيغة:

```
Select Table1.Column1, Table1.Column2, Table2.Column3
From Table1
Join Table2
ON Table1.Column1 = Table2.Column2;
```

مثال:
إذا كان لدينا جدول Names يحوي أسماء جميع الأشخاص (name) المقيمين في قرية مع أرقامهم التأمينية(INumber) والجدول Addresses الذي يتضمن عناوين المنازل (address) في تلك القرية مع الرقم التأميني (INumber) للمقيم في المنزل، وأردنا كتابة الاستعلام الذي يعيد قائمة بأسماء الأشخاص وعناوينهم، يمكننا استخدام الصيغة:

```
Select Names.name, Addresses.address from Names, Addresses
Where Names.INumber = Addresses.INumber;
```

أو

```
Select Names.name, Addresses.address
From Names
Join Addresses
ON Names.INumber = Addresses.INumber;
```

استخدام الربط بالتساوي مع عدة جداول

لا تتوقف إمكانيات الربط بالتساوي عند حدود الربط بين جدولين فقط بل يمكن أن تتعداها إلى مجموعة من الجداول.
نستخدم لإتمام هذه العملية الصيغة:

```
Select Table1.Column1, Table2.Column2, Table3.Column4
From Table1 Join Table2
ON Table1.Column1 = Table2.Column2
Join Table3
ON Table1.Column3 = Table3.Column4;
```

هنا يتم الربط بين الجداول اعتماداً على أسماء الحقول المتساوية المحددة بعد التعبير ON. ففي حالة الصيغة السابقة ربطنا الجدول Table1 مع Table2 معتمدين على تساوي قيم العمود Column1 من الجدول Table1 مع قيم العمود Column2 من الجدول Table2، وربطنا الجدول Table3 بالجدول Table1 معتمدين على تساوي قيم العمود Column3 من الجدول Table1 مع قيم العمود Column4 من الجدول Table3.

مثال:
لدينا الجدول Customers الحاوي على رمز الزبون customerID، اسمه customerName والجدول CreditCards الحاوي على رقم بطاقة الزبون الائتمانية cardNumber وعلى رقمه customerID، والجدول Addresses الحاوي على عناوين الزبائن وكان المطلوب إعادة قائمة برقم الزبون وأسمه، ورقم بطاقته الإئتمانية، والبلد الذي يقيم فيه country. للحصول على المطلوب، نكتب الاستعلام:

```
Select Customers.customerID, Customers.customerName,
CreditCards.cardNumber, Addresses.country
From Customers
Join CreditCards
ON Customers.customerID = CreditCards.customerID
Join Addresses
ON Customers.customerID = Addresses.customerID;
```

استخدام الربط بالتساوي مع عدة جداول (تتمة)

قد لا تعمل الصيغة السابقة على نظام إدارة قواعد البيانات MS Access لذلك قد يفضل البعض استخدام تقنية الربط المتساوي المتداخل أي استخدام الصيغة:

```
Select Table1.Column1, Table2.Column2, Table3.Column4
From Table2
Inner Join
(Table3 Inner Join Table1 ON Table3.Column4 = Table1.Column1)
ON Table2.Column2 = Table1.Column1;
```

تشبه هذه الصيغة تلك التي استخدمناها في الشريحة السابقة ولكن MS Access يواجه صعوبة في ترجمة عمليات الربط بصورة مباشرة، لذلك لجأنا إلى توليد بنى ربط متداخلة مجمعة بأقواس. فكما نلاحظ قمنا بربط الجدول Table3 مع الجدول Table1 بتساوي الحقلين Column4 من الجدول Table3 مع الحقل Column1 من الجدول Table1، ثم تعاملنا مع التعبير كاملاً كطرف في عملية الربط مع الجدول Table2 بتساوي الحقل Column2 من الجدول Table2 مع الحقل Column1 من الجدول Table1.

مثال:

إذا كان لدينا مخزن ألبسة يعتمد قاعدة بيانات تحتوي ثلاثة جداول: الأول جدول الأقسام Sectors (ولادي- نسائي- سن محير...)، الذي يحتوي على رمز القسم sectorID وإسم القسم sectorName. والثاني جدول الفصول Seasons الذي يحتوي رمز الفصول والسنوات (خريف 2004 – شتاء2003...) seasonID، وأسمائها seasonInfo. والثالث جدول المنتجات Products الذي يحتوي الأرقام التسلسلية للقطع productID، وتوصيفها productDescription، وأسعارها productPrice، ورمز القسم seasonID، ورمز القسم sectorID الذي تتبع له.

والمطلوب إظهار قائمة بتوصيف القطع وأسعارها، والأقسام التي تتبع لها والقسم الخاص بها.

الحل:

```
Select productDescription, productPrice, seasonName, sectorName
From Sectors
Inner Join
(Seasons Inner Join Products ON Seasons.seasonID = Products.seasonID)
ON Sectors.sectorID = Products.sectorID;
```

قد لا تعمل الصيغة السابقة على نظام إدارة قواعد البيانات MS Access لذلك قد يفضل البعض استخدام تقنية الربط المتساوي المتداخل. إذ يواجه MS Access صعوبة في ترجمة عمليات الربط بصورة مباشرة، لذلك نلجيء عادةً إلى توليد بنى ربط متداخلة مجمعة بأقواس.

الربط باللامساواة

يعتمد الربط بالمساواة على استخدام المساواة في شرط التعبير Where ولكن هذا لا يعني أننا لا نستطيع استخدام عمليات المقارنة الأخرى (أكبر، أصغر، وغيرها) كما في الصيغة التالية:

```
Select Table1.Column1, Table2.Column2
From Table1, Table2
Where Table1.Column1 < Table2.Column2;
```

مثال:

لدينا جدول Stores الخاص بمعمل أقمشة، والذي يحتوي يحتوي أسماء المخازن storeName وأرقامها storeID وجدول آخر Occupation يحتوي معلومات المشغولية للمخازن التي تتضمن رقم المخزن stroreID وكمية البضاعة في المخزن quantity ونوعها Type.

لإعادة قائمة بالمخازن الفارغة غير المشغولة نكتب الاستعلام:

```
Select  Stores.storeID, Stores.storeName from Stores, Occupation
Where Stores.storeID <> Occupation.storeID
```

الربط الخارجي

لنتمكن من فهم الربط الخارجي يجب أن نعود إلى فكرة الربط بالتساوي حيث استخدمنا التعبير Inner Join.

في حالة Inner Join، كانت السجلات التي أرجعها الاستعلام، هي السجلات التي تحقق شرط الربط الذي يظهر بعد تعبير ON، حيث تم اسقاط السجلات غير المتطابقة من جدول النتائج. أما في حالة الربط الخارجي Outer Join فلا يتم إسقاط السجلات غير المتطابقة.

للربط الخارجي ثلاثة أنواع: Left, Right, Full.

- لأخذ جميع السجلات من الجدول الأول Table1 وفقط السجلات من الجدول الثاني Table2 التي تتطابق فيها قيمة الحقل Column1 من الجدول Table1 مع قيمة الحقل Column2 من الجدول الثاني Table2، نكتب الصيغة:

```
Select * from Table1 LEFT OUTER JOIN Table2
ON Table1.Column1 = Table2.Column2;
```

- لأخذ جميع السجلات من الجدول الثاني Table2 وفقط السجلات من الجدول الأول Table1 التي تتطابق فيها قيمة الحقل Column1 من الجدول Table1 مع قيمة الحقل Column2 من الجدول الثاني Table2، نكتب الصيغة:

```
Select * from Table1 RIGHT OUTER JOIN Table2
ON Table1.Column1 = Table2.Column2;
```

- لأخذ جميع السجلات من الجدول الثاني Table2 وجميع السجلات من الجدول الأول Table1 بحيث تتوضع السجلات التي تتطابق فيها قيمة الحقل Column1 من الجدول Table1 مع قيمة الحقل Column2 من الجدول الثاني Table2 في نفس السجل من جدول القيم المعادة، نكتب الصيغة:

```
Select * from Table1 FULL OUTER JOIN Table2
ON Table1.Column1 = Table2.Column2;
```

ينتج عن عمليات الربط الخارجي، في الحالة العامة، سجلات تحتوي في حقول معينة القيمة NULL بسبب اختلاف عدد السجلات التي نريد ربطها، وهذا ما سنوضحه بالتفصيل لاحقاً مع مثال مناسب لكل نوع من أنواع الربط الخارجي.

الربط الخارجي

لنتمكن من فهم الربط الخارجي يجب أن نعود إلى فكرة الربط بالتساوي حيث استخدمنا التعبير Inner Join.

في حالة Inner Join، كانت السجلات التي أرجعها الاستعلام، هي السجلات التي تحقق شرط الربط الذي يظهر بعد تعبير ON، حيث تم اسقاط السجلات غير المتطابقة من جدول النتائج. أما في حالة الربط الخارجي Outer Join فلا يتم إسقاط السجلات غير المتطابقة.

للربط الخارجي ثلاثة أنواع: Left وRight وFull. ينتج عن عمليات الربط الخارجي، في الحالة العامة، سجلات تحتوي في حقول

معينة القيمة NULL بسبب اختلاف عدد السجلات التي نريد ربطها، وهذا ما سنوضحه بالتفصيل لاحقاً مع مثال مناسب لكل نوع من أنواع الربط الخارجي.

Left Join

لنستخدم صيغة الربط الخارجي التالية على الحقلين Column1 من الجدول الأول Table1، وColumn2 من الجدول الثاني Table2:

```
Select * from Table1 LEFT OUTER JOIN Table2
ON Table1.Column1 = Table2.Column2;
```

يُرجع هذا النوع من أنواع الربط الخارجي:

- جميع القيم الخاصة بالحقل التابع للجدول الأول،
- جميع القيم المطابقة لها والخاصة بالحقل التابع للجدول الثاني، حيث يتم إدراج القيمة Null في قيم الحقل التابع للجدول الثاني وذلك في السجلات التي لا تكون فيها قيمة حقل الجدول الأول مطابقة لقيمة حقل الجدول الثاني.

كما يكون عدد السجلات المعادة مطابقاً لعدد السجلات التي يعيدها الاستعلام عن قيم حقل الجدول الأول.

مثال:

لنستخدم الصيغة:

```
Select * from Table1 LEFT OUTER JOIN Table2
ON Table1.Column1 = Table2.Column2;
```

إذا كان لدينا جدول Table1 يحتوي في الحقل Column1 القيم {1, 5, 8, 3} والجدول Table2 الذي يحتوي في الحقل Column2 القيم {6, 5, 7, 9} فإن تطبيقها سيولد النتائج التالية:

Column1	Column2
1	Null
5	5
8	Null
3	Null

نلاحظ أنه تم إدراج القيمة Null لقيم الحقل Column2 في السجلات التي لم تكن فيها قيمة الحقل Column2 مطابقة لقيمة الحقل Column1.

نلاحظ أنه تم إدراج القيمة Null لقيم الحقل Column2 في السجلات التي لم تكن فيها قيمة الحقل Column2 مطابقة لقيمة الحقل Column1.

كما يكون عدد السجلات المعادة مطابقاً لعدد السجلات التي يعيدها الاستعلام عن قيم الحقل Column2 من الجدول Table2.

عندما نطبق هذا النوع من أنواع الربط الخارجي على حقلين من جدولين مختلفين، نحصل على:

- جميع القيم الخاصة بالحقل التابع للجدول الثاني،
- جميع القيم المطابقة لها والخاصة بالحقل التابع للجدول الأول، حيث يتم إدراج القيمة Null في قيم الحقل التابع للجدول الأول وذلك في السجلات التي لا تكون فيها قيمة حقل الجدول الثاني مطابقة لقيمة حقل الجدول الأول.

كما يكون عدد السجلات المعادة مطابقاً لعدد السجلات التي يعيدها الاستعلام عن قيم حقل الجدول الثاني.

Full Join

لنستخدم صيغة الربط الخارجي التالية على الحقلين Column1 من الجدول الأول Table1، وColumn2 من الجدول الثاني Table2:

```
Select * from Table1 FULL OUTER JOIN Table2
ON Table1.Column1 = Table2.Column2;
```

يُرجع هذا النوع من أنواع الربط الخارجي:

- جميع القيم الخاصة بالحقل التابع للجدول الأول، حيث يتم إدراج القيمة Null في قيم الحقل التابع للجدول الأول وذلك في السجلات التي لا تكون فيها قيمة حقل الجدول الثاني مطابقة لقيمة حقل الجدول الأول.
- جميع القيم المطابقة لها والخاصة بالحقل التابع للجدول الثاني، حيث يتم إدراج القيمة Null في قيم الحقل التابع للجدول الثاني وذلك في السجلات التي لا تكون فيها قيمة حقل الجدول الأول مطابقة لقيمة حقل الجدول الثاني.

مثال:

لنستخدم الصيغة:

```
Select * from Table1 FULL OUTER JOIN Table2
ON Table1.Column1 = Table2.Column2;
```

كما يكون عدد السجلات المعادة مطابقاً لعدد السجلات التي يعيدها الاستعلام عن قيم الحقل Column1 من الجدول Table1.

عندما نطبق هذا النوع من أنواع الربط الخارجي على حقلين من جدولين مختلفين، نحصل على:

- جميع القيم الخاصة بالحقل التابع للجدول الأول،
- جميع القيم المطابقة لها والخاصة بالحقل التابع للجدول الثاني، حيث يتم إدراج القيمة Null في قيم الحقل التابع للجدول الثاني وذلك في السجلات التي لا تكون فيها قيمة حقل الجدول الأول مطابقة لقيمة حقل الجدول الثاني.

كما يكون عدد السجلات المعادة مطابقاً لعدد السجلات التي يعيدها الاستعلام عن قيم حقل الجدول الأول.

Right Join

لنستخدم صيغة الربط الخارجي التالية على الحقلين Column1 من الجدول الأول Table1، وColumn2 من الجدول الثاني Table2:

```
Select * from Table1 RIGHT OUTER JOIN Table2
ON Table1.Column1 = Table2.Column2;
```

يُرجع هذا النوع من أنواع الربط الخارجي:

- جميع القيم الخاصة بالحقل التابع للجدول الثاني،
- جميع القيم المطابقة لها والخاصة بالحقل التابع للجدول الأول، حيث يتم إدراج القيمة Null في قيم الحقل التابع للجدول الأول وذلك في السجلات التي لا تكون فيها قيمة حقل الجدول الثاني مطابقة لقيمة حقل الجدول الأول.

كما يكون عدد السجلات المعادة مطابقاً لعدد السجلات التي يعيدها الاستعلام عن قيم حقل الجدول الثاني.

مثال:

لنستخدم الصيغة:

```
Select * from Table1 RIGHT OUTER JOIN Table2
ON Table1.Column1 = Table2.Column2;
```

إذا كان لدينا جدول Table1 يحتوي في الحقل Column1 القيم {1, 5, 8, 3} والجدول Table2 الذي يحتوي في الحقل Column2 القيم {6, 5, 7, 9} فإن تطبيقها سيولد النتائج التالية:

Column1	Column2
Null	6
5	5
Null	7
Null	9

إذا كان لدينا جدول Table1 يحتوي في الحقل Column1 القيم {1, 5, 8, 3} والجدول Table2 الذي يحتوي في الحقل Column2 القيم {6, 5, 7, 9} فإن تطبيقها سيولد النتائج التالية:

Column1	Column2
1	Null
5	5
8	Null
3	Null
Null	6
Null	7
Null	9

نلاحظ أنه تم إدراج القيمة Null لقيم الحقل Column1 في السجلات التي لم تكن فيها قيمة الحقل Column1 مطابقة لقيمة الحقل Column2 وإدراج القيمة Null لقيم الحقل Column2 التي لم تتطابق فيها قيمة Column1 مع Column2.

Full Join

عندما نطبق هذا النوع من أنواع الربط الخارجي على حقلين من جدولين مختلفين، نحصل على:

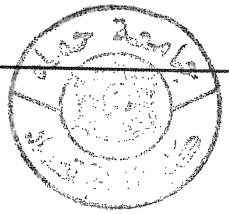
- جميع القيم الخاصة بالحقل التابع للجدول الأول، حيث يتم إدراج القيمة Null في قيم الحقل التابع للجدول الأول وذلك في السجلات التي لا تكون فيها قيمة حقل الجدول الثاني مطابقة لقيمة حقل الجدول الأول.
- جميع القيم المطابقة لها والخاصة بالحقل التابع للجدول الثاني، حيث يتم إدراج القيمة Null في قيم الحقل التابع للجدول الثاني وذلك في السجلات التي لا تكون فيها قيمة حقل الجدول الأول مطابقة لقيمة حقل الجدول الثاني.

استخدام Natural Join

يقوم التعبير Natural Join بعملية ربط اعتماداً على الحقول ذات الأسماء المُشتركة بين الجدولين والتي تحتوي على قيم متطابقة.
يأخذ التعبير الصيغة:

```
Select Table1.Column1, Table2.Column1 from Table1 Natural Join
Table2;
```

نلاحظ هنا أننا لم نستخدم التعبير ON وشرط التساوي لأنهما مُستخدمان ضمنياً في التعبير Natural Join حيث يتم تحقيق الربط عن طريق استخدام الحقل الذي يشترك الجدولان بإسمه.



مثال:

ليكن لدينا الجدول Pictures الحاوي على الأرقام التسلسلية للصور (pictureID) وعام تصويرها بالإضافة إلى شرح عن الصورة (pictureDescription). وليكن لدينا الجدول Names الحاوي على أسماء أصحاب الصور (clientName) والأرقام التسلسلية للصور (pictureID). فإذا كان المطلوب إعادة قائمة باسم كل شخص ووصف الصورة الخاصة به يمكننا كتابة الصيغة:

```
Select clientName, pictureDescription from Names Natural Join
Pictures;
```

نلاحظ في الصيغة السابقة أن الربط قد تم اعتماداً على الحقل الذي يشترك الجدولان بإسمه وهو pictureID.

استخدام Natural Join

يقوم التعبير Natural Join بعملية ربط لجميع الحقول ذات الأسماء المُشتركة والتي تحتوي على قيم متطابقة من الجدولين.
في هذا الاستعلام لا نستخدم التعبير ON وشرط التساوي لأنهما مُستخدمان ضمنياً في التعبير
Natural Join حيث يتم تحقيق الربط عن طريق استخدام الحقل الذي يشترك الجدولان بإسمه.

الربط باستخدام التعبير Using

يستخدم التعبير Using في حال كان اسم الحقل نعتمد على قيمه في عملية الربط متشابهاً في الجدولين.

يأخذ التعبير Using الصيغة:

```
Select Table1.Column2, Table2.Column3
From Table1 Join Table2 Using (Column1)
```

اعتبرنا هنا أن الربط يتم اعتماداً على قيم الحقل Column1 من الجدول الأول و Column1 من الجدول الثاني.

انتبه:

إن التعبيرين Using و Natural Join مدعومان من قواعد بيانات Oracle في نسختها 9I

يستخدم التعبير Using في حال كان اسم الحقل نعتمد على قيمه في عملية الربط متشابهاً في الجدولين.

التعامل مع أغراض قواعد البيانات

ركّزنا حتى الآن، وفي كل المواضيع التي تطرقنا إليها، على الاستعلام عن جداول في قاعدة بيانات معتبرين أن تلك قواعد البيانات منشأة مسبقاً والجداول موجودة. إلا أننا لم نتطرق إلى كيفية إنشاء قواعد البيانات: الجداول والفهارس والقيود عليها. وهذا ما يندرج تحت عنوان التعامل مع أغراض قواعد البيانات.

سنتعرف من خلال هذه الجلسة والجلسة القادمة على:

- كيفية إنشاء وحذف قاعدة بيانات.
- كيفية إنشاء وحذف وتعديل بنية الجداول.
- كيفية إنشاء وحذف وتعديل القيود على الحقول.
- كيفية إنشاء وحذف وتعديل الجداول المؤقتة وكيفية التعامل معها.
- كيفية إنشاء وحذف وتعديل الفهارس.
- كيفية إنشاء وحذف وتعديل المعاينات.

توليد وحذف قاعدة بيانات

تختلف طريقة تخزين وإدارة البيانات بين أنظمة إدارة قواعد البيانات. إلا أننا سنركز على التقنيات الأكثر شيوعاً.

* توليد قاعدة بيانات:

في أنظمة إدارة قواعد البيانات Oracle وSQL Server وMySQL وDB2 يمكنك ببساطة استخدام الصيغة:

```
CREATE DATABASE database_name;
```

* حذف قاعدة بيانات:

يمكننا حذف قاعدة بيانات ما باستخدام الصيغة:

```
DROP DATABASE database_name;
```

إلا أن التعبير (DROP DATABASE) مدعوم فقط من قواعد بيانات SQL Server وMy SQL وDB2، أما في Oracle فيمكننا استخدام نفس التوليد (CREATE DATABASE) لحذف قاعدة بيانات، كما يمكننا أن نستخدم تطبيق Oracle Database Assistant لحذف قاعدة بيانات.

انتبه:

- إن استخدام (CREATE DATABASE) مع إسم قاعدة بيانات موجودة في Oracle يؤدي إلى إلغاء واستبدال تلك القاعدة فتوخ الحذر عند استخدام هذا التعبير.
- تمتلك أنظمة إدارة قواعد البيانات آليات خاصة لتوليد قاعدة بيانات دون الحاجة إلى كتابة الصيغة الآنفة الذكر. فمثلاً يستخدم SQL Server تطبيق Enterprise Manager لتنفيذ عملية التوليد، وتستخدم قواعد بيانات DB2 تطبيق Control Center لنفس الغرض.
- لايدعم MS Access أياً من تعابير إنشاء وحذف قواعد البيانات لذلك نلجأ لاستخدام الخيار New من القائمة File في نافذة التطبيق Access. ولحذف قاعدة بيانات Access يكفي حذف ملف قاعدة البيانات تلك والذي يحمل اللاحقة (.mdb).

تختلف طريقة تخزين وإدارة البيانات بين أنظمة إدارة قواعد البيانات. إلا أننا سنركز على التقنيات الأكثر شيوعاً.

لتوليد قاعدة بيانات في أنظمة إدارة قواعد البيانات Oracle وSQL Server وMySQL وDB2 يمكنك ببساطة استخدام التعبير (CREATE DATABASE)

أما حذف قاعدة بيانات فيتم باستخدام التعبير (DROP DATABASE)

إلا أن التعبير (DROP DATABASE) مدعوم فقط من قواعد بيانات SQL Server وMy SQL وDB2، أما في Oracle فيمكننا استخدام نفس التوليد (CREATE DATABASE) لحذف قاعدة بيانات، كما يمكننا أن نستخدم تطبيق Oracle Database Assistant لحذف قاعدة بيانات.

إنشاء وحذف الجداول

لإنشاء جدول نستخدم الصيغة:

```
CREATE TABLE table_name
(column1_name column1_data_type column1_constraints,
column2_name column2_data_type column2_constraints,…);
```

لحذف جدول نستخدم الصيغة:

```
DROP TABLE table_name;
```

لتفريغ جميع السجلات من جدول ما نستخدم الصيغة:

```
TRUNCATE TABLE table_name;
```

مثال:

إذا أردنا إنشاء قاعدة بيانات Store وأردنا إنشاء جدولين ضمنها أحدهما باسم Customers والآخر باسم Products.

الجدول Customers يحوي الرقم التسلسلي للزبون (ID) واسم الزبون (name) ورقم هاتفه (phone). والجدول Products يحتوي الرقم التسلسلي (ID) لمنتج ووصف له (description).

لإنشاء هذه البنية نكتب الصيغة:

```
CREATE DATABASE Store;

CREATE TABLE Customers
(ID Int, name varchar(50), phone varchar(15);

CREATE TABLE Products
(ID Int, description varchar(75));
```

يمكننا بعد إنشاء الجداول البدء بإدراج سجلات ضمنها بالصيغة التي تعرفنا عليها في جلسات سابقة:

```
Insert into products (ID,description) values (1,'HPComputer');
```

نلاحظ هنا استخدامنا لأنواع البيانات Int وvarchar.

انتبه:

- لا يمكن عكس عملية حذف جدول من قاعدة البيانات في MySQL أي لا يمكن استرجاع الجدول بعد حذفه بعكس SQL Server وOracle وDB2 حيث يمكن التراجع عن الحذف واستعادة البيانات في حال تم إجراء تلك العملية كجزء من مناقلة.
- يمكننا في SQL Server, Oracle, MySQL إخلاء الجدول عوضاً عن حذفه أي تفريغ جميع السجلات، وهي أيضاً عملية غير عكوسة ولكنها أسرع من استخدام الصيغة:

```
DELETE from table_name
```

تستخدم التعابير (CREATE TABLE) و(DROP TABLE) و(TRUNCATE TABLE) لتوليد وحذف وإفراغ الجداول. وتوضح الصيغ الظاهرة كيفية استخدام كل منها.

نسخ الجداول

لنسخ جدول ما أثناء توليده يمكننا استخدام الصيغة:

```
CREATE TABLE table_name_copy AS Select* from table_name;
```

تتغير الصيغة بشكل طفيف في SQL Server وتصبح على الشكل:

```
Select * Into table_name_copy from table_name;
```

وفي MySQL تصبح الصيغة:

```
CREATE TABLE table_name_copy Select* from table_name;
```

في جميع الصيغ السابقة تمثل table_name إسم الجدول اذي نود نسخ بنيته مع البيانات الموجودة فيه وtable_name_copy الجدول المنسوخ عن الجدول table_name.

في حال أردنا نسخ بنية الجدول فقط بدون نسخ السجلات في الجدول يمكننا وضع شرط في تعبير Where له نتيجة False دائماً أي تصبح الصيغة على الشكل:

```
CREATE TABLE table_name_copy AS Select* from table_name
Where 1 = 0;
```

نلاحظ أن الشرط 1 = 0 لن يتحقق أبداً لذا سيتم نسخ بنية الجدول فقط ولن يوجد أي سجل سيحقق الشرط 1= 0.

أما في قواعد بيانات DB2 فسنضطر إلى إضافة التعبير DEFINITION ONLY فتصبح الصيغة من الشكل:

```
CREATE TABLE table_name_copy AS (Select* from table_name) DEFINITION
ONLY;
```

مثال:
لنسخ الجدول Logs إلى جدول آخر يسمى OldLogs نستخدم الصيغة:

```
CREATE TABLE OldLogs AS Select * from Logs;
```

لنسخ جدول ما أثناء توليده يمكننا استخدام تعبير (CREATE TABLE) أيضاً مع تغييرات طفيفة على صيغة الإستعلام بين Oracle و MySQL و SQLServer.

أما في حال أردنا نسخ بنية الجدول فقط بدون نسخ السجلات في الجدول فيمكننا وضع شرط في تعبير Where له نتيجة False دائماً، كوضع 0=1 على سبيل المثال.
وتشذ قواعد بيانات DB2 على ماسبق وتضطرنا لإضافة التعبير DEFINITION ONLY من أجل سخ بنية الجدول فقط دون سجلاتها.

تعديل بنية الجداول

لتعديل بنية جدول نستخدم الصيغة:

```
ALTER TABLE table_name [ADD | DROP COLUMN] (column_name [data_type] );
```

تساعد ADD على إضافة حقول جديدة، في حين تؤدي DROP إلى حذف حقول موجودة.

مثال:
لدينا الجدول Members الذي يحتوي الأرقام التسلسلية للأعضاء ID وأسماء الأعضاء Name.

يراد تعديل بنية الجدول بإضافة حقل جديد لإدخال نمط العضوية Type. لذا نكتب الصيغة:

```
ALTER TABLE Members ADD (Type varchar(15) );
```

لنفرض أننا نريد إلغاء الحقل ID من بنية الجدول Members. لتنفيذ ذلك نكتب الصيغة:

```
ALTER TABLE Members DROP COLUMN ID;
```

انتبه:
يساعد التعبير ALTER TABLE على إضافة أو إزالة قيود أو فهارس في بنية الجدول وهو ما سنعالجه لاحقاً في هذا الدرس.

لتعديل بنية جدول نستخدم التعبير (ALTER TABLE) الذي يمكننا من إضافة حقول أو حذف حقول من الجدول.

القيود على الحقول

لنعد مجدداً إلى الصيغة الخاصة بتوليد جدول:

```
CREATE TABLE table_name
(column1_name column1_data_type column1_constraints,
column2_name column2_data_type column2_constraints,…);
```

سنلاحظ أننا استخدمنا في هذه الصيغة التعبير column_constraints الذي يمثل القيد المراد تطبيقه على الحقل. سنتعرف فيما يلي على مجموعة من القيود على الحقول ونغطي كل منها بمثال مناسب.

من أهم القيود وأكثرها استخداماً:

- Not Null
- Default
- Primary key
- Unique
- Check
- Identity
- Auto_increment

انتبه:
يمكن لأكثر من قيد أن يطبق على نفس الحقل.

القيود على الحقول

إذا عدنا إلى الصيغة الخاصة بإنشاء جدول سنرى أننا استخدمنا التعبير column_ constraints الذي يمثل القيد المراد تطبيقه على الحقل. سنتعرف فيما يلي على مجموعة من القيود على الحقول ونغطي كل منها بمثال مناسب.

من أهم القيود وأكثرها استخداماً:

- Not Null
- Default
- Primary key
- Unique
- Check
- Identity
- Auto_increment

القيد NOT NULL

يستخدم هذا القيد في حال أردنا أن نمنع إدخال القيمة Null في الحقل المراد تقييده. يكفي لتطبيق هذا القيد إضافة التعبير NOT NULL عند إنشاء الجدول.

مثال:

إذا أردنا إنشاء جدول بأسماء الموظفين وتوصيف عملهم بحيث نمنع إعطاء القيمة Null لأي حقل من حقول الجدول نستخدم الصيغة:

```
CREATE TABLE Employees (name varchar(40) NOT NULL ,
Job varchar(50) NOT NULL);
```

إذا حاولنا تنفيذ الاستعلام التالي بغرض إضافة سجل إلى جدول الموظفين:

```
Insert into Employees(name) values('Adel')
```

سنلاحظ هذا التعبير لن يعمل وسيولد رسالة خطأ تفيد بضرورة إعطاء قيمة للحقل Job، كون القيمة التلقائية التي سيأخذها الحقل Job في حال عدم إدخال أي قيمة له هي القيمة Null.

القيد NOT NULL

يستخدم هذا القيد في حال أردنا أن نمنع إدخال القيمة Null في الحقل المراد تقييده. يكفي لتطبيق هذا القيد إضافة التعبير NOT NULL عند إنشاء الجدول.

تحديد قيمة تلقائية لحقل القيد DEFAULT

من القيود المستخدمة أيضاً بكثرة القيد الخاص بتحديد قيمة تلقائية لحقل ما. يأخذ هذا الحقل تلك القيمة حين لا يتم إسناد أية قيمة بديلة.

نستخدم القيد DEFAULT بالصيغة التالية:

```
CREATE TABLE MyTable
'(Column1 varchar(50) DEFAULT 'Unknown' ,
Column2 varchar(10) );
```

في هذه الصيغة سيتم إعطاء القيمة 'Unknown' للحقل Column1 في أي سجل جديد يتم إنشاؤه دون تحديد قيمة لـ . Column1.

مثال:
نريد توليد جدول على يحتوي وصف للبضائع (Description) وعدد الأيام الذي يلزم لنقلها إلى المستودع الرئيسي(Days) . نعلم مسبقاً أن عدد الأيام اللازم لعملية النقل هو يومان بصورة عامة.
لتوليد الجدول نكتب الصيغة:

```
CREATE TABLE Shipments
(Description varchar(75) Not Null , Days INT DEFAULT 2 Not Null);
```

نلاحظ أننا أنشأنا الجدول Shipments الذي يحتوي:

- الحقل Description ومنعنا إعطاء القيمة Null لهذا الحقل.
- الحقل Days من نمط بيانات الأعداد الصحيحة وحددنا القيمة 2 كقيمة تلقائية للحقل.

إذا حاولنا إدراج سجل ضمن هذا الجدول دون إعطاء قيمة لـ . Days كما في الصيغة:

```
INSERT INTO Shipments (Description) Values ('Computer');
```

سيكون شكل السجل الذي سيتم إدراجه في الجدول هو: Computer | 2
حيث أخذ الحقل Days في هذا السجل القيمة 2 علماً أننا لم نحدد هذه القيمة في صيغة إدراج السجل.

تحديد قيمة تلقائية لحقل القيد DEFAULT

يُعتبر القيد DEFAULT من القيود المستخدمة أيضاً بكثرة القيد الخاص بتحديد قيمة تلقائية لحقل ما. يأخذ هذا الحقل تلك القيمة حين لا يتم إسناد أية قيمة بديلة.

القيد PRIMARY KEY

من أهم الخصائص التي تميز الجداول في قواعد البيانات العلائقية والتي تعتبر من الشروط الأساسية للنموذج العلائقي لـ . Codd، ضرورة وجود حقل في كل جدول يلعب قيمه دور مميز لكل سجل من السجلات.

تعمل قيمة هذا المفتاح على تمييز كل سجل من سجلات الجدول بصورة وحيدة ويسمى المفتاح الرئيسي للجدول. لهذا الغرض يُستخدم القيد PRIMARY KEY ليحدد أي الحقول هو حقل مفتاح رئيسي لجدول ما.

يستخدم هذا القيد الصيغة:

```
CREATE TABLE MyTable
(Column1 data_type Not Null , Column2 data_type ,
Constraint myPrimaryKey PRIMARY KEY (Column1));
```

أنشأنا هنا قيد باسم MyPrimaryKey وقيدنا به الحقل Column1 بحيث يجب أن تكون قيمة هذا الحقل وحيدة وتميز كل سجل.

إذا لم نرد تحديد اسم لهذا القيد يمكننا كتابة الصيغة:

```
CREATE TABLE MyTable
(Column1 data_type Not Null , Column2 data_type ,
PRIMARY KEY (Column1));
```

أو بصورة أبسط يمكننا استخدام الصيغة:

```
CREATE TABLE MyTable
(Column1 data_type PRIMARY KEY Not Null , Column2 data_type ,
PRIMARY KEY (Column1));
```

مثال:

نريد إنشاء جدول CreditCards لتخزين أرقام البطاقات الائتمانية cardNumber وأسماء أصحابها cardHolder.

نعلم هنا أن أرقام البطاقات الائتمانية فريدة ولا تتكرر لذا سنعتمدها كمفتاح رئيسي في جدولنا وسنستخدم الصيغة المبسطة لقيد المفتاح الرئيسي:

```
CREATE TABLE CreditCards
(cardNumber varchar(20) PRIMARY KEY Not Null ,
cardHolder varchar(50) Not Null);
```

القيد Primary Key

من أهم الخصائص التي تميز الجداول في قواعد البيانات العلائقية والتي تعتبر من الشروط الأساسية للنموذج العلائقي لـ . Codd، ضرورة وجود حقل في كل جدول يلعب قيمه دور مميز لكل سجل من السجلات.

تعمل قيمة هذا المفتاح على تمييز كل سجل من سجلات الجدول بصورة وحيدة ويسمى المفتاح الرئيسي للجدول. لهذا الغرض يُستخدم القيد PRIMARY KEY ليحدد أي الحقول هو حقل مفتاح رئيسي لجدول ما.

القيد UNIQUE

يُستخدم القيد UNIQUE لمنع تكرار قيمة حقل في أكثر من سجل، ولكن هذا لايعني أن هذا الحقل سيصبح المفتاح الرئيسي للجدول.

الصيغة المستخدمة لهذا القيد هي كالتالي:

```
CREATE TABLE MYTable
(Column1 data_type UNIQUE , Column2 data_type);
```

مثال:

إذا أردنا توليد جدول PhoneBook بأسماء الأشخاص Name وأرقام هواتفهم Phone بحيث يكون رقم الهاتف مفتاح رئيسي واسم الشخص ذو قيمة وحيدة لا تتكرر نستخدم الصيغة:

```
CREATE TABLE PhoneBook
(Name varchar(50) UNIQUE , Phone Primary Key Not Null);
```

القيد UNIQUE

يُستخدم القيد UNIQUE لمنع تكرار قيمة حقل في أكثر من سجل، ولكن هذا لايعني أن هذا الحقل سيصبح المفتاح الرئيسي للجدول.

القيد Check

يعتبر القيد Check من أكثر القيود مرونة لأنه يسمح لنا باستخدام طيف واسع من الشروط على القيم التي يتم إدراجها ضمن الجدول.

تشبه طريقة استخدام شروط القيد Check لطريقة استخدام التعبير Where حيث يتم إعادة تقييم الشرط في كل مرة نضيف فيها سجلاً جديداً أو نعدل سجل موجود.

تكون صيغة استخدام القيد Check كالتالي:

```
CREATE TABLE MyTable
(Column1 data_type ,
 Column2 data_type ,
Constraint Cname CHECK (Condition));
```

حيث تعبر Cname عن اسم القيد وتعبر Condition عن شرط يُستخدم فيه اسم الحقل المراد تقييده بـ Check.

مثال:

نريد إنشاء جدول Ages يتضمن أسماء Name وأعمارAge مجموعة من الأطفال تتراوح بين سنة و12 سنة.

نكتب الصيغة:

```
CREATE TABLE Ages
(Name varchar(50) Not Null ,
 Age INT ,
Constraint CheckAge CHECK (Age between 1 And 12));
```

ويمكن أن نكتب صيغة تحقق نفس الغرض مع حذف اسم القيد CheckAge إذا كنا لانريد اسم للقيد.

```
CREATE TABLE Ages
(Name varchar(50) Not Null ,
Age INT CHECK(Age between 1 And 12));
```

يمكن لشرط القيد Check أن يتألف من تعبيرات منطقية تحتوي على عمليات منطقية مثل And أو Or فمثلاً إذا أردنا السماح بإدخال عمر بين 1 و12 أو يساوي 15 نكتب الصيغة:

```
CREATE TABLE Ages
(Name varchar(50) Not Null ,
Age INT CHECK(Age = 15 OR Age between 1 And 12));
```

يمكن وضع أكثر من قيد Check على حقل وحيد فمثلاً إذا أردنا السماح بالقيم للعمر بين 1 و12 عدا العمر 3 نكتب الصيغة:

```
CREATE TABLE Ages
(Name varchar(50) Not Null ,
Age INT,
Constraint CheckAge1 CHECK (Age between 1 And 12),
Constraint CheckAge2 CHECK (Age <> 3));
```

انتبه:

لا يطبق القيد Check عندما لايتلقى الحقل أي إدخالات (أي عندما تكون قيمته Null).

القيد Check

يعتبر القيد Check من أكثر القيود مرونة لأنه يسمح لنا باستخدام طيف واسع من الشروط على القيم التي يتم إدراجها ضمن الجدول.

- تشبه طريقة استخدام القيد Check طريقة استخدام التعبير Where حيث يتم إعادة تقييم الشرط في كل مرة نضيف فيها سجلاً جديداً أو نعدل سجل موجود.
- يمكن لشرط القيد Check أن يتألف من تعبيرات منطقية تحتوي على عمليات منطقية مثل And أو Or.
- يمكن وضع أكثر من قيد Check على حقل وحيد.

القيد IDENTITY و AUTO_INCREMENT

توفر قواعد البيانات آلية لتوليد قيم عددية بصورة آلية كقيم لحقل ما عند إضافة سجل جديد إلى الجدول. يمكن استخدام هذه التقنية بالاشتراك مع القيد PRIMARY KEY لتوليد قيم تسلسلية تلقائية في حقل يكون هو حقل المفتاح الرئيسي للجدول.

تستخدم قواعد البيانات تعابير مختلفة لتخديم نفس الغرض:

- نستخدم في SQL Server التعبير IDENTITY
- نستخدم في MySQL التعبير AUTO_INCREMENT
- نستخدم في DB2 التعبير GENRATED ALWAYS AS IDENTITY
- نستخدم في Access التعبير AUTOINCREMENT
- أما في Oracle فنحتاج لإنشاء متتالية بالأمر Create Sequence

مثال:

نود إنشاء جدول بأرقام تسلسلية للطلاب وأسمائهم:

- في قاعدة بيانات SQL Server:

الأرقام تبدأ من 100 وتتزايد بمقدار 1:

```
CREATE TABLE Students
(Name varchar(50) ,
ID INT IDENTITY (100,1) PRIMARY KEY NOT NULL);
```

في حال لم نحدد البذرة لبدء العد (100) والتزايد (1) تكون القيمة التلقائية للبذرة والتزايد هي (1).

- تصبح الصيغة في حالة MySQL بالشكل:

```
CREATE TABLE Students
(Name varchar(50) ,
ID INT AUTO_INCREMENT PRIMARY KEY NOT NULL);
```

- وتصبح الصيغة في Access:

```
CREATE TABLE Students
(Name varchar(50) ,
ID INT AUTOINCREMENT (100,1) PRIMARY KEY NOT NULL);
```

- أما في DB2 فتصبح الصيغة:

```
CREATE TABLE Students
(Name varchar(50) ,
ID INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY);
```

في هذه الصيغة نستخدم ALWAYS للإشارة إلى أننا لن نسمح بإدخال قيم يدوية لحقل التعداد الآلي. لتمكين إدخال القيم يدوياً يمكننا استخدام BY DEFAULT عوضاً عن ALWAYS.

القيد IDENTITY , AUTO_INCREMENT

توفر قواعد البيانات آلية لتوليد قيم عددية بصورة آلية كقيم لحقل ما عند إضافة سجل جديد إلى الجدول. يمكن استخدام هذه التقنية بالاشتراك مع القيد PRIMARY KEY لتوليد قيم تسلسلية تلقائية في حقل يكون هو حقل المفتاح الرئيسي للجدول.
تستخدم قواعد البيانات تعابير مختلفة لتخديم نفس الغرض:

- نستخدم في SQL Server التعبير IDENTITY
- نستخدم في MySQL التعبير AUTO_INCREMENT
- نستخدم في DB2 التعبير GENRATED ALWAYS AS IDENTITY
- نستخدم في Access التعبير AUTOINCREMENT
- أما في Oracle فنحتاج لإنشاء متتالية بالأمر Create Sequence

القيد IDENTITY وAUTO_INCREMENT

لا تدعم قواعد بيانات Oracle التعابير IDENTITY أو AUTO_INCREMENT المستخدمة في قواعد البيانات الأخرى. لذا تستخدم Oracle تعبير خاص لإنشاء متوالية هو التعبير CREATE SEQUENCE لمحاكاة عمل هذه القيود. تكون صيغة هذا التعبير على الشكل:

```
CREATE SEQUENCE sequence_name
INCREMENT increment_step
START WITH start_seed;
```

- عند إنشاء مثل هذه المتوالية يمكن استخدام القيم التي تأخذها لإدراجها في صيغ SQL أخرى.
- لإعادة قيمة المتوالية وزيادة عداد المتوالية بمقدار الخطوة يكفي استخدام التعبير sequence_name.NextVal

يتم استخدام قيم هذه المتوالية مباشرة في تعبير Insert عند إدراج سجل جديد في جدول ما وذلك بالصيغة:

```
INSERT INTO mytable
(Column1, Column2, Column3)
Values (sequence_name.NextVal, Value2, Value3);
```

نلاحظ هنا أننا أعطينا للحقل Column1 القيمة الخاصة بالمتوالية واستخدمنا الطريقة NextVal لاستجرار القيمة التالية للمتوالية.

مثال:

نريد إنشاء جدول Products في قاعدة بيانات Oracle يحتوي حقلين الأول خاص بأرقام المنتجات productID والثاني وصف المنتجات ProductDescription. كما نود جعل الحقل productID حقل مفتاح رئيسي وجعل قاعدة البيانات تعطي لهذا الحقل أرقام متسلسلة تلقائية.

لإتمام هذا العمل نقوم أولاً بإنشاء الجدول بالصيغة:

```
CREATE TABLE Products
(productID INT PRIMARY KEY NOT NULL ,
productDescription varchar(75));
```

ثم نقوم بإنشاء متوالية وليكن اسمها مثلاً Counter وذلك بالصيغة:

```
CREATE SEQUENCE Counter;
```

حيث لم نحدد هنا بداية العد والخطوة لذلك ستحدد قيمتها تلقائياً بالعدد 1.

عند إدراج سجل في جدولنا الجديد يجب استخدام المتوالية كقيمة للحقل productID وذلك بالشكل:

```
INSERT INTO Products
(productID , productDescription)
Values (Counter.NextVal , 'any Porduct description');
```

لا تدعم قواعد بيانات Oracle التعابير IDENTITY أو AUTO_INCREMENT المستخدمة في قواعد البيانات الأخرى. لذا تستخدم Oracle تعبير خاص لإنشاء متوالية هو التعبير CREATE SEQUENCE لمحاكاة عمل هذه القيود.

عند إنشاء مثل هذه المتوالية يمكن استخدام القيم التي تأخذها لإدراجها في صيغ SQL أخرى.
لإعادة قيمة المتوالية وزيادة عداد المتوالية بمقدار خطوة يكفي استخدام التعبير sequence_name.NextVal

التكامل المرجعي 1

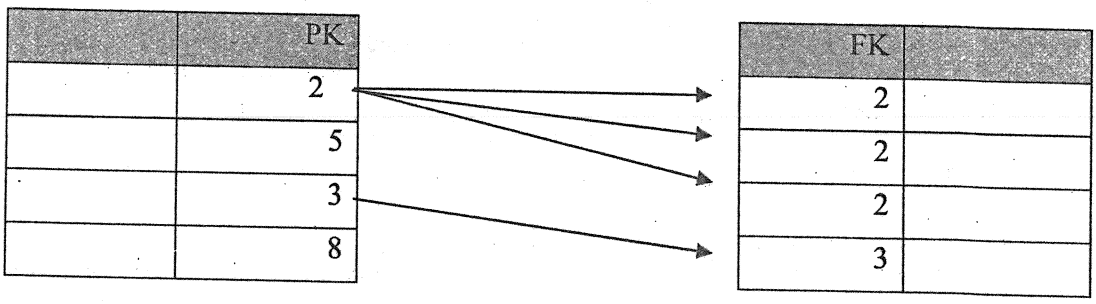
تتألف قواعد البيانات العلائقية من مجموعة من الجداول يتشكل كل من هذه الجداول من مجموعة من السجلات. يميز كل سجل من سجلات جدول قيمة فريدة لحقل أطلقنا عليه اسم حقل المفتاح الرئيسي.

قد ترتبط الجداول بعضها ببعض بعلاقات يمكن لهذه العلاقات أن تأخذ أحد الأشكال:

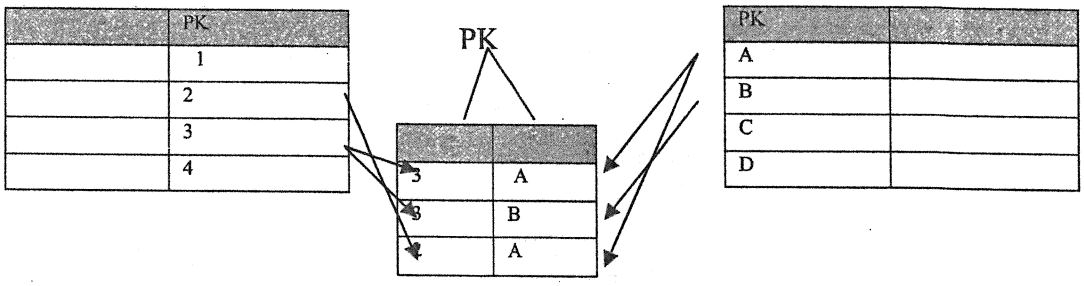
- علاقة واحد لواحد أو سجل لسجل: في هذا النوع من العلاقات يرتبط كل سجل من الجدول الأول مع سجل وحيد من الجدول الثاني.
- علاقة واحد لعديد أو سجل لعدة سجلات: يرتبط كل سجل من الجدول الأول مع مجموعة سجلات من الجدول الثاني وليس العكس.
- علاقة عديد لعديد أو عدة سجلات لعدة سجلات: ترتبط مجموعة من سجلات الجدول الأول مع مجموعة من سجلات الجدول الثاني.

التكامل المرجعي 2

علاقة سجل لعدة سجلات: يتم عادة التعبير عن العلاقة بإدراج ما يسمى المفتاح الثانوي في حقل خاص في الجدول الثاني (الذي يتوضع في الجهة التي توجد فيها مجموعة السجلات المرتبطة بالسجل الوحيد) وبحيث تكون قيم هذا الحقل مأخوذة من قيم حقل المفتاح الرئيسي للجدول الأول.



أما في علاقة عدة سجلات لعدة سجلات فهي تطبّق فعلياً على شكل علاقتين من نوع سجل لعدة سجلات مع استخدام جدول وسيط يحتوي مفتاح أساسي مركب يتكون من قيمة المفتاح الأساسي للسجل المراد ربطه من الجدول الأول مع قيمة المفتاح الأساسي للسجل المراد ربطه من الجدول الثاني.



التكامل المرجعي 2

علاقة سجل لعدة سجلات: يتم عادة التعبير عن العلاقة بإدراج ما يسمى المفتاح الثانوي في حقل خاص في الجدول الثاني (الذي يتوضع في الجهة التي توجد فيها مجموعة السجلات المرتبطة بالسجل الوحيد) بحيث تكون قيم هذا الحقل مأخوذة من قيم حقل المفتاح الرئيسي للجدول الأول.

أما في علاقة عدة سجلات لعدة سجلات فهي تطبّق فعلياً على شكل علاقتين من نوع سجل لعدة سجلات مع استخدام جدول وسيط يحتوي مفتاح أساسي مركب يتكون من قيمة المفتاح الأساسي للسجل المراد ربطه من الجدول الأول مع قيمة المفتاح الأساسي للسجل المراد ربطه من الجدول الثاني.

التكامل المرجعي 3

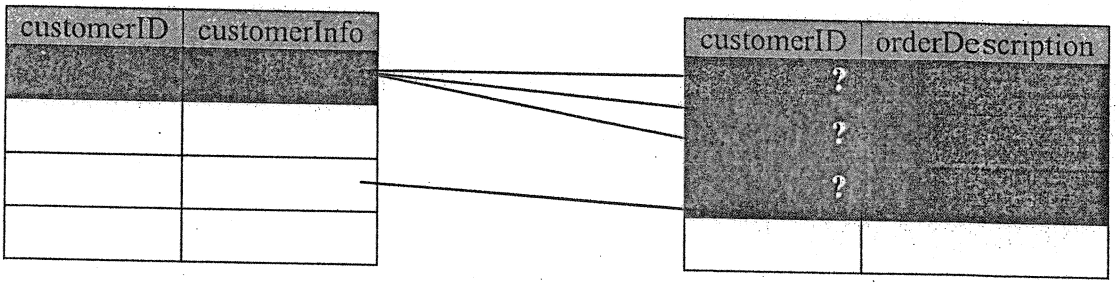
لنفرض أن لدينا قاعدة بيانات علائقية مؤلفة من جدولين يرتبط الأول بالثاني بعلاقة واحد لعديد (سجل من الأول بعدة سجلات من الثاني). سبق وأسلفنا أن طريقة حلّ العلاقة، تتلخص في إدراج مفتاح في الجدول الثاني يسمى المفتاح الثانوي مصدره قيم المفتاح الرئيسي للجدول الأول.

لكن ماذا لو اضطررنا إلى حذف أو تعديل أحد السجلات من الجدول الأول المرتبطة بعدد من السجلات من الجدول الثاني ؟

فعلياً ما سنحصل عليه هو قيم جديدة غير مترابطة ببعضها البعض. وهنا تكمن أهمية المحافظة على ماندعوه التكامل المرجعي في قاعدة البيانات.

مثال:

لنفرض أن لدينا جدول يحتوي معلومات الزبائن وجدول آخر يحتوي طلبات الشراء التي قام بها الزبائن. من الواضح أن العلاقة بين الجدولين هي علاقة سجل من جدول الزبائن لعدة سجلات من جدول الطلبات. فإذا خطر بذهن أحدهم حذف اسم أحد الزبائن من جدول الزبائن وكان لهذا سجلات طلبات في جدول الطلبات فسنحصل بعد عملية الحذف تلك على طلبات في جدول الطلبات لا نعلم الزبون الذي طلبها.



كما نلاحظ في المخطط أعلاه أن حذف السجل من الجدول Customers (إلى اليسار) سيؤدي إلى ضياع مرجعية السجلات في الجدول Orders (إلى اليمين).

التكامل المرجعي 3

لنفرض أن لدينا قاعدة بيانات علائقية مؤلفة من جدولين يرتبط الأول بالثاني بعلاقة واحد لعديد (سجل من الأول بعدة سجلات من الثاني). سبق وأسلفنا أن طريقة حلّ العلاقة، تتلخص في إدراج مفتاح في الجدول الثاني يسمى المفتاح الثانوي مصدره قيم المفتاح الرئيسي للجدول الأول.

لكن ماذا لو اضطررنا إلى حذف أو تعديل أحد السجلات من الجدول الأول المرتبطة بعدد من السجلات من الجدول الثاني ؟

فعلياً ما سنحصل عليه هو قيم جديدة غير مترابطة ببعضها البعض. وهنا تكمن أهمية المحافظة على ماندعوه التكامل المرجعي في قاعدة البيانات.

فإذا فرضنا أن لدينا جدول يحتوي معلومات الزبائن وجدول آخر يحتوي طلبات الشراء التي قام بها الزبائن. من الواضح أن العلاقة بين الجدولين هي علاقة سجل من جدول الزبائن لعدة سجلات من جدول الطلبات. فإذا خطر بذهن أحدهم حذف اسم أحد الزبائن من جدول الزبائن وكان لهذا سجلات طلبات في جدول الطلبات فسنحصل بعد عملية الحذف تلك على طلبات في جدول الطلبات لا نعلم الزبون الذي طلبها.

التكامل المرجعي 4

لضمان عدم كسر قاعدة التكامل المرجعي في SQL، علينا استخدام قيد يعرّف أحد الحقول في جدول على أنه حقل مفتاح ثانوي لجدول آخر بحيث يفشل أي تعبير لا يحترم قاعدة التكامل المرجعي بعد استخدام هذا القيد.

يتم لهذا الغرض استخدام الصيغة:

```
CREATE TABLE myTable
(Column1 Column1Type PRIMARY KEY NOT NULL , Column2 Column2Type ,
Column3 Column3Type ,
CONSTAINT foreign_key_name FOREIGN KEY (Column3)
REFERENCES other_table (other_table_primary_key));
```

نلاحظ في الصيغة السابقة أننا أنشأنا الجدول myTable الذي يحوي الحقول Column1 كحقل مفتاح رئيسي والحقل Column2 والحقل Column3 الذي تم تقييده بالقيد FOREIGN KEY لجعله مفتاح ثانوي مرتبط بالمفتاح الرئيسي للجدول other_table والذي يحمل اسم other_table_primary_key.

مثال:

إذا أردنا إنشاء الجدول Brands الذي يحتوي أسماء وأرقام ماركات أجهزة الحواسب المتوفرة بمتجر، والجدول Models الذي يحتوي الموديلات المتوفرة من كل ماركة من الماركات، وأردنا ربط الجدول Brands مع الجدول Models بعلاقة سجل من جدول Brands إلى عدة سجلات من جدول Models مع ضمان التكامل المرجعي نكتب الصيغة التالية لإنشاء الجدول Brands:

```
brandID INT PRIMARY KEY NOT NULL ,      CREATE TABLE Brands (
brandName varchar (50));
```

لإنشاء الجدول Models نستخدم الصيغة:

```
modelID INT PRIMARY KEY NOT NULL ,      CREATE TABLE Models (
modelName varchar (50) ,
modelBrand INT ,
CONSTRAINT myFK FOREIGN KEY (modelBrand)
REFERENCES Brands (brandID));
```

هذه الصيغة صالحة في SQL Server, Oracle, DB2 و Access.

يمكن كتابة هذه الصيغة بشكل أكثر اختصاراً إذا كنا لا ننوي تحديد اسم القيد بالشكل:

```
modelID INT PRIMARY KEY NOT NULL ,     CREATE TABLE Models (
modelName varchar (50) ,
modelBrand INT ,
FOREIGN KEY (modelBrand)
REFERENCES Brands (brandID));
```

التكامل المرجعي 4

لضمان عدم كسر قاعدة التكامل المرجعي في SQL، علينا استخدام قيد يعرّف أحد الحقول في جدول على أنه حقل مفتاح ثانوي لجدول آخر بحيث يفشل أي تعبير لا يحترم قاعدة التكامل المرجعي بعد استخدام هذا القيد.

التكامل المرجعي في MySQL

في MySQL تختلف الصيغة بنقطتين أساسيتين:

الأولى هي أن الجداول الداخلة في العلاقة يجب أن تكون من نوع الجداول الخاص في قواعد بيانات MySQL وهو InnoDB، والثانية هي ضرورة تعريف حقل المفتاح الثانوي كفهرس فتصبح الصيغة من الشكل:

```
CREATE TABLE myTable
(Column1 Column1Type PRIMARY KEY NOT NULL , Column2 Column2Type ,
Column3 Column3Type ,
FOREIGN KEY (Column3)
REFERENCES other_table (other_table_primary_key)
INDEX myIndex (Column3))
Type = InnoDB;
```

التكامل المرجعي في MySQL

في MySQL تختلف الصيغة بنقطتين أساسيتين:
الأولى هي أن الجداول الداخلة في العلاقة يجب أن تكون من نوع الجداول الخاص في قواعد بيانات MySQL وهو InnoDB
والثانية هي ضرورة تعريف حقل المفتاح الثانوي كفهرس.

نموذج (نوع) البيانات	طبيعة المدخلات	الحجم التخزيني
Text نص	نص أو تركيبة نصوص وأرقام، كالعناوين. وكذلك الأرقام التي لا تتطلب حسابات، كأرقام الهاتف، أو أرقام الأجزاء،	حتى ٢٥٥ حرفاً. ١بايت لكل حرف.
Currency العملة	قيم العملة. استخدم نوع البيانات "عملة" لمنع حدوث التقريب أثناء إجراء الحسابات. بالضبط ويصل عدد الخانات إلى ١٥ خانة إلى يسار الفاصلة العشرية و ٤ خانات إلى يمينها.	8بايت.
Number رقم	تستخدام البيانات الرقمية للحسابات الرياضية، باستثناء الحسابات المتعلقة بالأموال (استخدام نوع العملة).	1، 2، 4، أو ٨ بايت
Date/Time الوقت التاريخ	تاريخ وزمن .	8بايت.
نعم/لا Yes/No	حقول تحتوي فقط على قيمة واحدة أو اثنتين، مثل "نعم/لا"، و"صحيح/خطأ"، و"تشغيل/إيقاف".	1بايت.
AuoNumber ترقيم تلقائي	الأرقام الفريدة المتتالية (التي تزيد بمقدار ١) أو الأرقام العشوائية يتم إدراجها تلقائياً عند إضافة سجل.	4بايت. ١٦ بايت لـ "معرف"
Memo مذكرة	نص أو أرقام طولية، كالملاحظات أو الوصف.	حتى ٦٤.٠٠٠ حرفاً.
OLE Object كائن OLE	البرامج (مثل مستنداتMicrosoft Word ، أو جداول بياناتMicrosoft Excel ، أو صور، أو أصوات، أو أي بيانات ثنائية أخرى)، التي تم إنشاؤها في برامج أخرى باستخدام البروتوكولOLE ، والتي يمكن ربطها بجدول في Microsoft Accessأو تضمينها فيه.	حتى ١ جيجا بايت (مقيدة بواسطة مساحة القرص).

أنواع البيانات في MS - Access

Hyperlink ارتباط تشعبي	حقول سوف تقوم بتخزين ارتباطات تشعبية يمكن أن يكون الارتباط التشعبي مسار UNC أو URL.	حتى ٦٤.٠٠٠ حرفاً.
Lookup معالج البحث	يقوم بإنشاء حقل يسمح لك باختيار قيمة من جدول آخر أو من قائمة قيم باستخدام مربع تحرير وسرد. يؤدي اختيار هذا الخيار في قائمة أنواع البيانات إلى بدء معالج لتعريف هذا النوع من أجلك.	تفس حجم الحقل المفتاح الأساسي والذي هو أيضاً الحقل "بحث"؛ ٤ بايت بالضبط

الجدول (٣-١) نماذج البيانات في MS-ACESS