Debreceni Egyetem Informatikai Kar

Szakdolgozat

Készítette: Berecz Károly

Debrecen, 2007.

Debreceni Egyetem Informatikai Kar

Szakdolgozat

Adatbázis-alapú alkalmazás-fejlesztés Borland Delphiben

Készítette: Berecz Károly
Programozó Matematikus Szak

Témavezető: Dr. Bajalinov Erik
Alkalmazott Matematika
és Valószínűségszámítás Tanszék

Tartalomjegyzék

I. Bevezetés

Az általános felhasználói réteget tekintve állíthatjuk, hogy a PC-k világát éljük. A számítógép megjelenésével az emberi munka hatékonyságának növelése és különböző munkaterületek gépesítése volt a cél. A gyors és dinamikus fejlődés miatt mára már az élet minden területén megjelent a számítógép úgy, mint a médiában, szórakozásban, tudományban és a hétköznapi munkában. Előtérbe került a hatékonyság és az adatbázis-kezelők megjelenése lehetővé tette nem csak az adatok számítógépen való tárolását, hanem feldolgozását is. Igényként jelennek meg a grafikus felületű, a felhasználó minden igényét kielégítő és esztétikus programok. A grafikus felületek szorosan kapcsolódnak a grafikus operációs rendszerekhez, amik magukban hordoznak olyan eszközrendszert, amely megkönnyíti mind a fejlesztők, mind a felhasználók munkáját.

A szakdolgozatomban egy anyagigénylő, igénylés nyilvántartó alkalmazást készítettem Delphi fejlesztői környezetben, mely a gyakorlatban is alkalmazható. A feladat olyan alkalmazás készítése volt, ami a vasúti anyag vételezéseket rögzíti. Az anyagvételezések egyrészt vontatójármű karbantartásra, javításra, üzemeltetésre, másrészt vontatott jármű karbantartásra, javításra, üzemeltetésre történnek. Ezeken felül még megjelenik a dolgozók kéziszerszámainak, védőeszközeinek, védőitalainak a vételezése is.

Az igény az volt, hogy a különböző telephelyeken végrehajtott vételezések nyomon követhetők legyenek, akár dolgozóként, akár telephelyenként, akár Fenntartási Központ szinten is. Szükséges igényként jelentkezett a kiírt tételek nyomtatványra történő gépi rögzítése is.

Az alkalmazásban rögzítésre kerül többek között az anyagot kiíró dolgozó, az anyag megnevezése, cikkszáma, a vételezett mennyiség, vételezés dátuma, a felhasználási projekt, munkaszám, és nem utolsósorban az utalvány száma. Ezen adatok segítségével az igényként jelentkező kimutatások elkészíthetők.

A program hardverszükséglete közepesnek mondható, különböző Windows alapú operációsrendszereken futtatható, a nyomtatáshoz bármilyen Windows alá telepített nyomtató megfelelő.

II. Adatbázis-tervezés

Múlt és jelen

A múltban az adatbázisok tervezése az informatikai szakemberek és a hivatásos adatbázis- fejlesztők feladata volt. Ezek az emberek általában matematikai, számítástechnikai vagy rendszertervezői háttérrel rendelkeztek, és többnyire nagygépes adatbázisrendszerekkel dolgoztak. Sokan közülük tapasztalt programozók voltak, akik számos, több ezer sorból álló adatbázis-alkalmazás fejlesztését tudhatták maguk mögött. (Ezek az emberek munkájuk természetéből és fontosságából adódóan rendszerint igen túlterheltek voltak.)

Akkoriban az adatbázisok tervezése szilárd oktatási hátteret igényelt, mivel e rendszerek nagy részét egy teljes vállalat használta. Még abban az esetben is, ha csak egy részleg vagy egy kisebb cég számára kellett adatbázist tervezni, a tervezők felkészítése sokrétű oktatást igényelt, mivel meglehetősen összetett programozási nyelveket és adatbázis-alkalmazásokat kellett használniuk. Az eszközök fejlődésével azonban ezek a követelmények megváltoztak.

Az 1980-as évek közepétől kezdve sok szoftvergyártó foglalkozott asztali gépeken futtatható adatbázisprogramok fejlesztésével. Ezen eszközök programozása sokkal egyszerűbb volt, mint nagygépeken futó társaikéi. Megjelentek azok a programok is, amelyek lehetővé tették, hogy emberek egy csoportja központosított adatokat osszon meg egymással a helyi hálózat ügyfél-kiszolgáló rendszerein, nagyterületű hálózatokon vagy akár az Interneten keresztül. A cégek és a különféle szervezetek nem függtek többé a nagygépes adatbázisoktól, adatigényeik kielégítéséhez pedig nem volt szükség központi IT részlegekre. Az évek során a gyártók sok új szolgáltatással egészítették ki eszközeiket, így egyre hatékonyabb és egyre rugalmasabb adatbázis-alkalmazások fejlődtek ki. Lényegesen leegyszerűsödött a programok használata is, ami egyre inkább arra ösztönözte az embereket, hogy maguk tervezzék meg adatbázisaikat. A mai adatbázis-alkalmazások hihetetlen mértékben leegyszerűsítik a hatékony adatbázis-szerkezetek és a hozzájuk kapcsolódó felhasználói felületek létrehozásának folyamatát.

A legtöbb program mintaadatbázisokat tartalmaz, amelyeket lemásolhatunk, és saját igényeinkre szabhatunk. Bár első ránézésre igen előnyösnek tűnik, hogy saját adatbázisainkat ezek alapján készítsük el érdemes megállni és elgondolkozni egy kicsit. Miért? Azért, mert ezzel a módszerrel könnyen előfordulhat, hogy a létrejövő adatbázisterv nem lesz elég hatékony vagy éppen nem lesz teljes, és végül komoly gondokba ütközhetünk az addig

megbízhatónak hitt adatbázisunkban. Ez természetesen felveti a kérdést: milyen gondokról van szó?

A legtöbb probléma, amellyel egy adatbázisban találkozhatunk, két csoportra osztható: alkalmazásbeli problémák és adatproblémák. Alkalmazásbeli probléma lehet például az adatbevitel nehézkessége, a menük átláthatatlansága, a párbeszédablakok zavarossága és a fárasztó munkafolyamatok szükségessége. Ezek a gondok általában akkor jönnek elő, ha a tervező nem elég tapasztalt, nincs tisztában a jó alkalmazás-tervezés módszereivel, vagy nem ismeri eléggé az adatbázis megvalósításához használt programot.

Az adatokkal kapcsolatos problémák között megemlíthetjük a hiányzó, a helytelen, az egymáshoz nem illeszkedő és a pontatlan adatokat. Az ilyen típusú gondok hátterében általában a nem megfelelő adatbázis-tervezés áll. Az adatbázis nem fogja tudni kielégíteni a szervezet igényeit, ha nem megfelelő a szerkezete.

Fejlesztés

Az adatbázis-fejlesztés folyamatát három szakaszra bonthatjuk:

1. **Logikai tervezés.** Az első szakaszhoz tartozik a táblák és a mezők meghatározása, az elsődleges és az idegen kulcsok kiválasztása, a táblák közötti kapcsolatok leírása és az adatépség különböző szintjeinek megvalósítása.
2. **Fizikai megvalósítás.** A második szakasz a táblák létrehozását, a kulcsmezők és a táblák közötti kapcsolatok kialakítását és az adatépség különböző szintjeinek megvalósításához szükséges eszközök használatát jelenti.
3. **Alkalmazásfejlesztés.** A harmadik szakaszban egy alkalmazást hozunk létre, amely hozzáférést biztosít a felhasználó (vagy felhasználók) számára az adatbázisban tárolt adatokhoz. Az alkalmazásfejlesztési szakaszt különféle folyamatokra bonthatjuk. Ilyen többek között a végfelhasználói feladatok azonosítása, a jelentésekben feltüntetni kívánt adatok összegyűjtése és az alkalmazás bejárásához szükséges menürendszer létrehozása.

Először mindig a logikai tervet kell elkészítenünk, lehetőleg minél teljesebben. Miután kialakítottuk a szerkezetet, bármilyen adatbázis-szoftverrel létrehozhatjuk azt. A megvalósítás szakaszába lépve elképzelhető, hogy a választott adatbázisprogram tulajdonságaiból kifolyólag módosítanunk kell az adatbázis szerkezetét. Még az is lehet, hogy az adatfeldolgozás teljesítményének növelése miatt döntünk az adatbázis-szerkezet megváltoztatása

mellett. A logikai terv elkészítése során tudatosan, módszeresen és megalapozott döntésektől vezérelve alakítjuk ki az adatbázis szerkezetét. Ennek eredményeképpen kicsire csökken annak a valószínűsége, hogy a fizikai megvalósítás és az alkalmazásfejlesztés szakaszában kell majd megváltoztatnunk azt.

Az adatbázis egy szervezet vagy egy szervezeti folyamat modellezésére szolgáló adatok rendezett gyűjteménye. Lényegtelen, hogy adatokat papíron vagy számítógépen tároljuk. Mindaddig, amíg szervezett módon gyűjtjük az adatokat, adatbázisról beszélünk.

Korai adatbázismodellek

A relációs adatbázismodell megjelenése előtt az adatok tárolására két különböző modellt használtak: a *hierarchikus* és a *hálós adatbázismodellt*.

1. A hierarchikus adatbázismodell

Az adatokat ebben a modellben hierarchikus módon tároljuk, és egy felfelé fordított fával ábrázolhatjuk. A fa gyökere ennek az adatbázisnak egy táblája, a többi tábla pedig a gyökérből kiinduló ágak mentén helyezkedik el.

A hierarchikus adatbázis *szülő-gyermek* típusú kapcsolatokból épül fel. Ebben a kapcsolatban a szülőtáblához egy vagy több gyermektábla tartozhat, a gyermektábláknak azonban csak egyetlen szülőjük lehet. A táblákat a mezők fizikai elrendezésével vagy mutatókkal kapcsoljuk össze. A felhasználó az adatok olvasását mindig a gyökérből kezdi, és innen halad végig a kívánt adatokig.

E modell használatának előnye, hogy a felhasználó gyorsan hozzájut az adatokhoz, illetve automatikusan megvalósul a hivatkozási épség. Biztosak lehetünk abban , hogy gyermektábla egy rekordja a szülőtábla egy létező rekordjához kapcsolódik, és abban is, hogy a szülőtáblából történő törlés automatikusan kihat a gyermektáblákra is.
Gondot okoz azonban, amikor olyan rekordot szeretnénk elhelyezni a gyermektáblában, amely jelenleg a szülőtábla egyetlen rekordjához sem kapcsolódik. Ez az adatbázistípus nem alkalmas összetett kapcsolatok leírására, és gyakran problémát jelentenek az ismétlődő adatok.

A hierarchikus adatbázisok megfelelő megoldásnak bizonyultak a szalagos tároló-eszközöket használó nagygépek esetében, és nagyon népszerűnek számítottak az 1970-es években.

2. A hálós adatbázismodell

Ezt a modellt elsősorban a hierarchikus adatbázisok egyes problémáinak megoldására fejlesztették ki. A hálós adatbázis szerkezetét a csomópont és halmazszerkezet fogalmakkal írhatjuk le. A csomópontok a rekordegyütteseket, a halmazszerkezetek pedig a hálós adatbázisban fennálló kapcsolatokat jelképezik. Ez a szerkezet két csomópont, a tulajdonos és a tag között teremt kapcsolatot. A halmazszerkezet támogatja az egy-sok kapcsolatokat is, ami azt jelenti, hogy a tulajdonos csomópont egy rekordjához a tag csomópontból több rekord is tartozhat, a tag csomópont egy rekordjához azonban a tulajdonos rekordjai közül szigorúan csak egy tartozik. Ezen kívül a tag csomópontban nem létezhet olyan rekord, amely nem kapcsolódik a tulajdonos valamelyik rekordjához. Két csomópont között egy vagy több kapcsolatot is meghatározhatunk, és a csomópontok más csomópontokkal is kapcsolatban lehetnek.

A hálós adatbázisban tárolt adatok eléréséhez nem kell végigjárnunk az összes kapcsolatot. A hierarchikus adatbázissal ellentétben, ahol mindig a gyökértáblától kell indulnunk, a felhasználó a hálós adatbázis bármelyik csomópontjától elindulhat, és innen tetszőleges irányban mozoghat a kapcsolatok mentén.

A hálós adatbázis egyik előnye az adatok gyors elérhetősége. Ezen kívül ez az adatbázistípus a hierarchikus adatbázisoknál megszokottaknál összetettebb lekérdezések összeállítását teszi lehetővé. A hálós adatbázis fő hátránya, hogy a felhasználónak pontosan ismernie kell az adatbázis felépítését ahhoz, hogy bejárhassa a benne lévő kapcsolatokat.

3. A relációs adatbázismodell

A relációs adatbázis ötlete 1969-ben született meg, és azóta a legszélesebb körben használt adatbázismodellé nőtte ki magát. A relációs modell szülőatyja, Dr. Edgar F. Codd, az 1960-as évek végén kutatóként dolgozott az IBM-nél, és a nagy adatmennyiségek tárolásának új módszereit kereste. Az akkori adatbázismodellekkel és adatbázistermékekkel való elégedetlensége arra késztette, hogy matematika alapelveinek felhasználásával próbálja megoldani a felmerülő temérdek problémát. Mivel hivatásos matematikus volt, erősen hitt abban, hogy a matematika bizonyos ágai megoldást adnak az olyan problémákra, mint amilyen az adatok ismétlődése, a gyenge adatépség, és az adatbázis szerkezetének a fizikai megvalósítástól való erős függősége.

Dr. Codd új relációs modelljét 1970. júniusában mutatta be. Az új modellt a matematika két ágára, a halmazelméletre és az elsőrendű predikátumkalkulusra építette fel. A modell neve a reláció szóból ered, amely egy halmazelméleti fogalom.

A relációs adatbázis az adatokat kapcsolatokban tárolja, amelyek a felhasználó számára táblák formájában jelennek meg. A kapcsolatok sorokból (rekordokból) és jellemzőkből (mezőkből) állnak. A rekordok és a mezők fizikai sorrendje a táblában teljesen lényegtelen. A tábla rekordjait egy egyedi értékeket tartalmazó mező azonosítja. A relációs adatbázisoknak ez a két tulajdonsága teszi lehetővé, hogy az adatok létezése teljesen független legyen azok fizikai tárolásától.

A relációs modell háromféle kapcsolatot különböztet meg: egy-egy, egy-sok és sok-sok.

A relációs adatbázisban tárolt adatokhoz az SQL (Structured Query Language) segítségével férhetünk hozzá. Az SQL a relációs adatbázisok létrehozására, módosítására, karbantartására és lekérdezésére alkalmas szabványos nyelv.
Egy egyszerű SQL lekérdezés egy SELECT. . . FROM utasításból, egy WHERE záradékból és egy ORDER BY záradékból áll. A SELECT záradékban soroljuk fel a lekérdezésben használni kívánt mezőket, a FROM záradék pedig azokat a táblákat adja meg, amelyek ezeket a mezőket tartalmazzák. A lekérdezésben visszakapott rekordokat a WHERE záradékban megadott feltételekkel szűrhetjük meg, majd az így kapott végeredményt az ORDER BY záradékban meghatározott módon rendezhetjük.

A relációs adatbázisok számos előnyt biztosítanak a korábbi modellekkel szemben:

- *Beépített többszintű épség*. A mezők szintjén megjelenő adatépség biztosítja az adatok pontosságát; a táblaszintű épség gondoskodik arról, hogy a rekordok ne jelenhessenek meg kétszer a táblában és arról is, hogy ne hiányozzon az elsődleges kulcs; a kapcsolat szintjén megjelenő épség garantálja a táblák közötti kapcsolatok érvényességét; végül a működési szint épsége biztosítja, hogy az adatok a működési (üzleti) szabályoknak is megfeleljenek.
- *Az adatok logikai és fizikai függetlensége az adatbázis-alkalmazásoktól.* Sem az adatbázis logikai szerkezetének változása, sem pedig az adatbázis fizikai megvalósítása nincs hatással az adatbázisra épülő alkalmazásokra.
- *Az adatok garantált következetessége és pontossága.* Az adatbázis különböző szintjein jelenlévő épség biztosítja az adatok következetességét és pontosságát.
- *Az adatok egyszerű lekérdezhetősége.* A felhasználó parancsára könnyedén hozzájuthatunk egy tábla vagy egymással összekapcsolt táblák adataihoz. Ennek köszönhetően a felhasználó lehetőségei az adatok lekérdezésére szinte korlátlanok.

A relációs adatbázis-kezelő rendszer (RDBMS) olyan szoftver, amely alkalmas relációs adatbázisok létrehozására, módosítására és kezelésére. A legtöbb RDBMS program olyan eszközöket is biztosít számunkra, amelyekre az adatbázisban tárolt adatokkal dolgozó végfelhasználói alkalmazásoknak szükségük van.

Az 1980-as évek első felében fellendült a személyi számítógépek elterjedése, és ezzel egyidőben megjelentek a PC-re írt RDBMS programok. A kategória első képviselői között említhetjük a dBase-t (Ashton-Tate) és a FoxPro-t (Fox Software), amelyek egyszerű fájlalapú adatbázis-kezelő rendszerek voltak.

Az 1980-as évek végén és az 1990-es évek elején, miközben egyre több felhasználó kezdett adatbázisokat használni, nyilvánvalóvá vált az adatok megosztásának szükségessége. A sok felhasználó számára elérhető, központosított adatbázis ötlete ígéretesen csengett. A központosítás megkönnyítette volna az adatok kezelését és biztonságossá tételét. A gyártók az ügyfél-kiszolgáló RDBMS programok kifejlesztésével válaszoltak erre az igényre.

Az RDBMS programokat az általános üzleti alkalmazásokban (például leltárnyilvántartás, ügyfélkezelés, megrendelés-feldolgozás, eseménynaptárak) széles körben használják, léteznek azonban olyan alkalmazások is (például számítógépes tervezés, földrajzi információs rendszerek, multimédiás tárolórendszerek), amelyek kiszolgálására nem alkalmasak.
A probléma megoldására két adatbázismodell jelent meg: az objektum orientált adatbázis és az objektum-relációs adatbázis.

Az adatbázisok felhasználásának módja hatalmasat változott az elmúlt néhány év során. Eljött az idő, amikor egyre több és több vállalat kezdte felismerni, hogy rengeteg olyan adat van, amelyeket össze lehetne gyűjteni egy relációs vagy más típusú adatbázisban. Ezen a ponton végig kellett gondolniuk, hogy össze tudják-e gyűjteni valamilyen módszerrel azokat az adatokat, amelyek az üzleti döntések meghozatalában segíthetnek. Sőt, azt is tudni szerették volna, hogyan építhetnek ezekből az adatokból egy életképes tudásbázist szervezetük számára.

Az Internetnek nagy szerepe volt abban, hogy a különféle szervezetek adatbázisokat kezdtek használni. Sok cég vásárlóinak számát szeretné növelni a világhálón való jelenlétével, és a honlapokon keresztül megosztott adatok nagy része adatbázisokból származik. Az Internet életképes megoldást teremtett a különféle relációs és nem relációs adatbázisok összeolvasztására is. Az XML (eXtensible Markup Language; bővíthető jelölőnyelv) rövid idő alatt a heterogén rendszerek közötti adattovábbítás szabványává vált. Az XML

nem függ a használt rendszertől, így az adatbázisrendszer minden további gond nélkül képes írni és olvasni a más rendszerekből érkező XML dokumentumokat.

Adatbázis-tervezés.

Az adatbázis-szerkezet megtervezésébe fektetett idő jócskán megtérül. A jó tervezéssel hosszú távon időt takaríthatunk meg, mivel nem kell állandóan a gyorsan és rosszul megtervezett szerkezetet javítgatni. A jó tervezés az alábbi előnyöket biztosítja számunkra:

- *Az adatbázis szerkezete könnyen módosítható és karbantartható lesz.* A mezők és a táblák módosítása nem érinti az adatbázis többi mezőjének vagy táblájának szerkezetét.
- *Az adatok módosítása könnyen elvégezhető.* Egy mező értékének megváltoztatása nincs hatással a többi mező értékére. Sőt, egy Jól megtervezett adatbázisban nagyon kevés kétszer szereplő mező van, így az adott értéket csak egy mezőben kell megváltoztatnunk.
- *Könnyű kinyerni az adatokat.* A lekérdezések összeállítása egyszerű feladat, mivel jó a táblák szerkezete és megfelelően fel vannak építve a táblák közötti kapcsolatok.
- *A végfelhasználói alkalmazások fejlesztése egyszerű feladat.* Több időnk marad a programozásra, és az adatkezeléssel kapcsolatos feladatok elvégzésére, mivel nem kell állandóan az adatbázis rossz felépítéséből fakadó hibákkal foglalkoznunk.

Az adatmodellezés fázisában meghatározzuk a mezőket is és hozzárendeljük azokat a megfelelő táblákhoz. Minden táblához tartozik egy *elsődleges kulcs,* a táblák az adatok épségének különféle szintjeit valósítják meg, a táblák közötti kapcsolatokról pedig idegen kulcsok gondoskodnak. Miután a kezdeti táblaszerkezet elkészült, és kialakítottuk az adatmodellnek megfelelő kapcsolatokat, nekiláthatunk az adatbázis normalizálásának.
A *normalizálás* során a nagy táblákat kisebb táblákra bontjuk, hogy kiszűrjük ezzel az ismétlődő adatokat, és elkerüljük az adatok beszúrása, frissítése és törlése során felmerülő gondokat. A normalizálás során a táblaszerkezeteket *normálformákkal* vetjük össze, majd módosítjuk azokat, ha a fenti problémák valamelyikébe ütközünk. A normálforma olyan szabályhalmaz, melynek segítségével ellenőrizhetjük, hogy a tábla szerkezete egészséges és problémamentes-e. Jelenleg a következő normálformákat használhatjuk: első normál-

forma, második normálforma, harmadik normálforma, negyedik normálforma, ötödik normálforma, Boyce-Codd normálforma.

A tervezési folyamat néhány fogalma

A fogalmakat négy csoportba oszthatjuk: *értékekkel kapcsolatos fogalmak, szerkezetekkel kapcsolatos fogalmak, kapcsolatokkal kapcsolatos fogalmak* és az *épséggel kapcsolatos fogalmak.*

Értékekkel kapcsolatos fogalmak

Adat

Az adatbázisban tárolt értékeket adatoknak nevezzük. Az adat statikus abban az értelemben, hogy változatlan állapotban van mindaddig, amíg valamilyen kézi vagy automatikus folyamat révén meg nem változtatjuk. Az adatnak önmagában nincs jelentése.

Információ

Az *információ* olyan adat, amely a feldolgozás révén értelmet nyer, azaz tudjuk, mit jelent, amikor látjuk vagy éppen dolgozunk vele. Az információ dinamikus abban az értelemben, hogy jelentése az adatbázisban tárolt adatok függvényében folyamatosan változik, és abban az értelemben is, hogy számtalan módon feldolgozhatjuk és megjeleníthetjük. A SELECT utasítás eredményét megjeleníthetjük például a képernyőn, de jelentés formájában ki is nyomtathatjuk. A lényeg, hogy *az adatok csak akkor válnak információvá, ha értelmes módon dolgozzuk fel azokat.*

Az *adat* az, amit *tárolunk;* az *információ* pedig, amit *kinyerünk.*

Null

A *null* hiányzó vagy ismeretlen értéket jelöl. Kezdettől fogva tisztában kell lennünk azzal, hogy a *null* nem azonos a nullával, és nem azonos az egy vagy több szóközből álló karakterlánccal sem.

Szerkezetekkel kapcsolatos fogalmak

Tábla

A relációs modell szerint a relációs adatbázis adatait kapcsolatokban (relációkban) tároljuk, amelyek a felhasználó számára *táblák* formájában jelennek meg. Minden kapcsolat sorokból (rekordokból) és *jellemzőkből* (mezőkből) áll.

A táblák az adatbázis legfontosabb szerkezetei; minden tábla egyetlen, jól meghatározott tárgyat ír le. A rekordok és a mezők sorrendje teljesen lényegtelen, és minden tábla tartal-

maz legalább egy mezőt *(elsődleges kulcs),* amely egyedi módon azonosítja a tábla rekordjait.

Azokat a táblákat, amelyek a tárolt adatok alapján információt szolgáltatnak, *adattáblák*nak nevezzük; a relációs adatbázisokban legtöbbször ilyen típusú táblákkal találkozunk. Az ilyen táblákban tárolt adatok általában dinamikusak, mivel módosíthatjuk és többféle módon feldolgozhatjuk azokat.

Mező

A *mező* az adatbázis legkisebb szerkezete, amely a tábla tárgyának egy jellemzőjét írja le. A mezők tárolják a tényleges adatokat. A mezőkben tárolt adatokhoz azután hozzáférhetünk, és számtalan módon megjeleníthetjük azokat.

Rekord

A *rekord* (amelyet a relációs adatbázisok elmélete *sornak* nevez) a tábla tárgyának egy egyedi példányát írja le. A rekord a teljes mezőkészletet magában foglalja, függetlenül attól, hogy az adott mezők tartalmaznak-e értékeket.

Nézettábla

A *nézettábla* egy virtuális tábla, amely az adatbázis egy vagy több táblájának mezőiből áll össze; a nézettáblát felépítő táblákat együttesen *alaptábláknak* nevezzük. A relációs modell virtuálisnak nevezi ezeket a táblákat, mivel nem tárolnak adatokat, hanem tartalmukat más táblákból nyerik. Az adatbázis kizárólag a szerkezetét tartalmazza.

A nézettáblák jelentőségének három fő oka van.

1. Lehetővé teszik, hogy több tábla adataival dolgozhassunk egyszerre. (Ahhoz, hogy ezt megtehessük, a tábláknak *kapcsolódniuk* kell egymáshoz.)
2. Megakadályozhatjuk, hogy bizonyos felhasználók hozzáférhessenek egy tábla vagy egy táblacsoport adott mezőihez. Ez a lehetőség biztonsági szempontból nagyon előnyös.
3. Biztosíthatjuk a segítségükkel az adatok épségét. Az ilyen célra használt nézettáblákat *érvényesítő nézettábláknak* nevezzük.

Kulcsok

A kulcsok olyan mezők, amelyek különleges szerepet töltenek be a táblában; a kulcs szerepét a típusa határozza meg. Egy tábla számos különféle típusú kulcsot tartalmazhat, a legfontosabbak azonban az *elsődleges* és a *másodlagos kulcsok.*

Az elsődleges kulcs olyan mező vagy mezőcsoport, amely egyedi módon azonosítja a rekordokat a táblán belül; ha az elsődleges kulcs több mezőből áll, akkor *összetett* elsődleges kulcsnak nevezzük. Az elsődleges kulcs a tábla legfontosabb kulcsa.

Amikor kapcsolatot teremtünk két tábla között, általában fogjuk az egyik tábla elsődleges kulcsát, és beépítjük azt a második tábla szerkezetébe, ahol ez az érték idegen kulccsá válik. Az idegen kulcs kifejezés abból ered, hogy a második táblának is van elsődleges kulcsa, így az első táblából átvett elsődleges kulcs „idegen" a második tábla számára.

Index

Az *index* olyan szerkezet, amellyel az RDBMS az adatok feldolgozásának hatékonyságát igyekszik javítani.

Kapcsolatokkal kapcsolatos fogalmak

Kapcsolatok

Ha két tábla rekordjai valamilyen értelemben összetartoznak, akkor azt mondjuk, hogy a két tábla kapcsolódik egymáshoz. A táblák közötti kapcsolatokat elsődleges és másodlagos kulcsok, illetve kapcsoló tábla segítségével alakíthatjuk ki.

Kapcsolattípusok

A táblák között lévő kapcsolatoknak három különféle típusa létezik: *egy-egy, egy-sok* és *sok-sok.*

Egy-egy kapcsolatok

Két tábla között *egy-egy* kapcsolatról beszélünk, ha az első tábla egy rekordjához a második táblából csak egy rekord tartozik, és a második tábla minden rekordja az első táblában csak egy rekordnak feleltethető meg.

Egy-sok kapcsolatok

Egy-sok kapcsolatról beszélünk két tábla között, ha az első tábla egy rekordjához a második táblában több rekord is tartozhat, a második tábla egy rekordjának azonban az első táblában csak egy rekord felel meg.

Sok-sok kapcsolatok

Két tábla között sok-sok kapcsolatról beszélünk, amennyiben az első tábla egy rekordjához a második táblából több rekord is tartozhat, és a második tábla egy rekordja az első táblából több rekorddal is összekapcsolható.

A részvétel típusai

A tábla részvétele a kapcsolatban lehet *kötelező vagy nem kötelező* Tegyük fel, hogy A és B két, egymással kapcsolatban álló tábla.

- Az A tábla részvétele a kapcsolatban kötelező, ha az A táblának legalább egy rekordot tartalmaznia kell, mielőtt a B táblába felvennénk az első rekordot.

- Az A tábla részvétele a kapcsolatban nem kötelező, amennyiben a B táblába úgy is felvehetünk rekordokat, hogy az A tábla még egyet sem tartalmaz.

A részvétel mértéke

A *részvétel mértéke* határozza meg azt a legkisebb és legnagyobb rekordszámot, amellyel egy adott tábla a vele kapcsolatban álló tábla egy rekordjához kapcsolódhat.
Tegyük fel, hogy A és B két, egymással kapcsolatban álló tábla. A kapcsolat mértékének meghatározásához meg kell mondanunk, hogy legalább hány B-beli rekordnak kell kapcsolódnia és legfeljebb hány B-beli rekord kapcsolódhat A *egyetlen* rekordjához.

Az épséggel kapcsolatos fogalmak

Mező-meghatározás

A *mező-meghatározás* (hagyományos nevén *tartomány* egy mező összetevőit írja le.
A mező-meghatározások a jellemzők három típusát tartalmazzák: *általános, fizikai* és *logikai.*

- Az *általános* jellemzők szolgáltatják a mezőről az alapvető információkat. Ezek tartalmazzák a mező nevét, leírását és a szülőtábla nevét.
- *A fizikai* jellemzők határozzák meg, hogyan épül fel a mező, és hogyan jelenik meg a mezőt használó személy számára. Ebbe a csoportba tartozik az adattípus, a méret és a megjelenítési formátum.
- A *logikai* jellemzők a mezőben tárolt értékeket írják le; itt találjuk meg a kötelező érték, az értéktartomány és az alapértelmezett érték beállításokat.

Adatépség

Az *adatok épsége* az adatbázisban tárolt adatok érvényességére, következetességére és pontosságára vonatkozó fogalom.
Az adatbázis tervezése folyamán négyféle adatépséget kell megvalósítanunk. Ezek közül három az adatbázis szerkezetén alapul, és ennek megfelelően kapta a nevét is. Az adatok épségének negyedik típusa azon alapszik, ahogyan a szervezet az adatokat használja.

1. A *táblaszintű épség (egyedépség)* biztosítja, hogy ne kerülhessen be ugyanaz a rekord kétszer a táblába, és hogy legyen olyan mező, amely egyedi módon azonosítja a rekordokat.
2. A *mezőszintű épség (tartományépség)* gondoskodik arról, hogy minden mező hibamentes legyen; a mezők értékei érvényesek, következetesek és pontosak lesznek; az azonos típusú mezők meghatározása az egész adatbázisban ugyanolyan.

3. A *kapcsolatszintű épség (hivatkozási épség)* szavatolja, hogy a táblák közötti kapcsolat hibátlan legyen, és hogy az adatok beszúrása, módosítása és törlése esetén a különböző táblák rekordjai összhangban maradjanak egymással.
4. A *működési szabályok* („üzleti szabályok") a szervezet adathasználatából adódóan korlátozzák az adatbázis tartalmát. Ezek a megszorítások sokfélék lehetnek; meghatározhatják például egy mező lehetséges értékeit, a táblák részvételének típusát és mértékét a különféle kapcsolatokban, és a kapcsolatszintű épség összehangolásának módját.

Az adatbázis tervezés lépései

Célok és részfeladatok az adatbázis tervezésben

Az adatbázis-tervezés folyamatának első lépése az adatbázissal kapcsolatos *célok* és *rész feladatok* kijelölése. A célok meghatározásával *(„küldetési leírás")* pontosan leírhatjuk, mire is szeretnénk használni az adatbázist, így behatárolhatjuk tervezési munkánk területét.

A célok mellett meg kell határoznunk a részfeladatokat *(„küldetési célokat")* is - ezek mentén tesszük lehetővé a felhasználók adatkezelési műveleteit.

Adatbázis-elemzés

Az adatbázis-tervezés folyamatának második lépése a meglevő adatbázis elemzése - már amennyiben létezik ilyen. Ez lehet *„hagyaték"* vagy *papír alapú adatbázis.* Előbbi esetben egy olyan adatbázisról van szó, melyet a cég már évek óta használ, míg utóbbi esetben az „adatbázis" gyakorlatilag űrlapok, mutatók, mappák és más hasonlók gyűjteményéből áll össze.

Az elemzés következő része interjúkból áll össze, melyekben a felhasználókat és a vezetőség tagjait faggatjuk arról, hogyan dolgoznak nap mint nap az adatbázissal. A felhasználókat meg kell kérdeznünk arról, miként dolgoznak az adatbázissal, és jelenleg milyen adatok elérését tartanák fontosnak. A vezetőséget viszont arról kell kérdeznünk, milyen adatok jutnak el hozzájuk, és milyen képük alakult ki a cég általános adatfelhasználásáról.

Adatszerkezetek összeállítása

Az adatbázis-tervezés folyamatának harmadik szakasza az adatszerkezetek létrehozása. Itt határozzuk meg a táblákat és a mezőket, a kulcsokat és a mezők tulajdonságait.

Táblák közti kapcsolatok feltérképezése és kiépítése

Az adatbázis-tervezés folyamatának negyedik szakasza a táblák közti kapcsolatok kiépítése. Itt ismét interjúkat kell készítenünk a felhasználókkal és a vezetőséggel, azonosítva a kapcsolatokat, azok jellemzőit, és így biztosítva a kapcsolatszintű adatépséget.

Működési szabályok meghatározása és létrehozása

A tervezési folyamat következő szakasza a működési szabályok meghatározása. A korábbiakhoz hasonlóan itt is interjúkat kell készítenünk, majd meg kell határoznunk az adatbázis különböző területein alkalmazandó korlátozásokat, ki kell dolgoznunk a működési szabályokat, és meg kell adnunk az érvényesítő táblákat.

Nézettáblák meghatározása és létrehozása

A tervezési folyamat hatodik szakasza a nézettáblák meghatározása, amihez megint csak interjúkat kell készítenünk.

Ahhoz, hogy azonosítsuk az adatbázisba beépítendő nézettáblák típusait, el kell beszélgetnünk a felhasználókkal és a vezetőséggel, hogy megtudjuk, miként is dolgoznak az adatokkal. Kiderülhet például, hogy sok felhasználónak részletes adatokra van szüksége a munkához, míg mások összefoglaló tájékoztatást igényelnek a cég életét meghatározó stratégiai döntésekhez. A felhasználók egyes csoportjai más és más szemszögből néznek az adatokra. a nézettáblák helyes meghatározásával hathatós segítséget nyújthatunk számunkra.

A következő feladat a nézettáblák elkészítése az interjúkban elhangzottak alapján, ehhez meg kell határoznunk a megfelelő táblákat és mezőket, valamint össze kell állítanunk az egyedi adatokat szolgáltató nézettáblák feltételeit.

Az adatépség ellenőrzése

Az adatbázis-tervezés folyamatának hetedik, végső szakaszában az eddigiekben elkészített adatbázis-szerkezetet vizsgáljuk meg, számon kérve rajta az adatépség követelményeit.

Először az egyes táblákat vesszük górcső alá, ellenőrizve szerkezetüket, valamint mezőik kapcsolatait. Feloldjuk az esetleges ellentmondásokat, kiküszöböljük a felfedezett hibákat, majd még egyszer átnézzük az egész táblát. Ha ezzel elkészültünk, ellenőrizzük az adatok épségét a tábla szintjén.

Másodszor, megvizsgáljuk a mezők jellemzőit. Elvégezzük a mezők szükséges módosításait, majd ellenőrizzük a mezőszintű adatépséget. Ezzel egyúttal megerősítjük az adatok a s korábban biztosított mezőszintű épségét.

Harmadszor a kapcsolatok érvényességét, típusát, valamint a kapcsolatok és a táblák viszonyát ellenőrizzük. Ezután megvizsgáljuk az adatok épségét a kapcsolatok szintjén - meggyőződünk róla, hogy a közös mezőkben azonos értékek szerepelnek, és hogy a beillesztés frissítés és törlés végbemegy az adatépség sérülése nélkül, bármelyik, a kapcsolatban részt vevő táblán is végezzük.
Végezetül, ellenőrizzük a korábban rögzített működési szabályokat, és újfent meggyőződünk a megszorítások szükségességéről. Ha az utolsó interjúk óta újabb korlátozások váltak szükségessé, most ezeket is beépíthetjük az adatbázisba, működési szabályok formájában.

A tervezési folyamat végén az eddigiekben felépített logikai szerkezetet egy relációs adatbázis-kezelő program segítségével a gyakorlatban is megvalósítjuk.

III. A Delphi rendszer bemutatása

Object Pascal

A Delphi rendszert elsősorban a Windows rendszer alatt futó, a grafikus felhasználói felület lehetőségeit kihasználó alkalmazások készítésére használjuk. A Delphi eseményvezérelt nyelv, egy negyedik generációs fejlesztőeszköz, melynek nyelve az Object Pascal. Az OO technika lehetővé teszi a program kódjának és adatainak összefonódását, magas szinten megvalósítva az absztrakciót. Az objektumokkal és az objektumok hierarchiájának felhasználásával egyszerű szerkezetű programok készíthetők.

Az Object Pascal nyelv a Turbo Pascal nyelv továbbfejlesztett objektum orientált változatának tekinthető, mely tartalmazza az objektum orientált programozás alapelemeit, az egységbefoglalást (encapsulation), az öröklődést (inheritance), és a polimorfizmust (polymorphism).

A bezárást, egységbefoglalást (encapsulation) az osztályok valósítják meg. Az Object Pascal nyelvben az objektumok típusát osztálynak (Class) nevezzük. Az objektumok adatainak hozzáférési jogainak jellemzésére használhatjuk a nyilvános (public), a saját (private), a védett (protected) kulcsszavakat. A privát láthatóságú adatokat, jellemzőket és metódusokat csak az illető osztály metódusaiból érhetők el. Ha a láthatóság védett, akkor csak az adott osztályból és az ebből származtatott osztályok metódusaiban láthatóak az adatok, jellemzők és metódusok. Nyilvános adatok más objektumok metódusai számára is elérhetővé válnak.

Az osztály egy felhasználó által definiált adattípus, aminek van állapota (a megjelenése) és vannak műveletei (a viselkedése).
Az osztálynak vannak belső adatai és metódusai, eljárások és függvények képében. Az objektum az osztály egy példánya, vagy más szóval egy, az adott osztály által definiált típusú változó. Az objektumok valójában egyedek, és amikor a program fut, akkor az objektumok memóriát foglalnak le a belső értékeik számára. Egy osztálynak akármennyi adata és akárhány metódusa lehet.

Öröklődés alapja, hogy létezik egy olyan osztály, amiből az összes többi osztály származik. Ez az előre definiált TObject osztály, ami minden osztály közös őse és a Vizuális Komponens Könyvtár (VCL) alapját képezi. Egy új osztályt közvetlenül egy régi, már meglévő alapján definiálunk. A származtatott osztály örökli az alaposztály tulajdonságait (viselkedését, mezőit), amelyek szükségszerűen meg is változtathatók, újak definiálhatók,

illetve a meglévők felüldefiniálhatók. A közös ős azt teszi lehetővé, hogy annak adattípusával bármely, a Delphiben előforduló típust helyettesíthessünk.

A Delpiben dinamikus kötés, más néven késői kötés van. Ez azt jelenti, hogy futásidőben a metódushívás pontos címét a hívást kezdeményező példány típusa határozza meg. A típus kompatibilitás miatt a Delphi csak futásidőben tudja meghatározni a változóban tárolt objektum tényleges típusát. Ha a metódus szerepel az osztály deklarációjában, akkor egyértelmű, hogy az hajtódik végre, ha nem, akkor az öröklődési gráfban kell visszafelé keresni, hogy hol történt a deklaráció

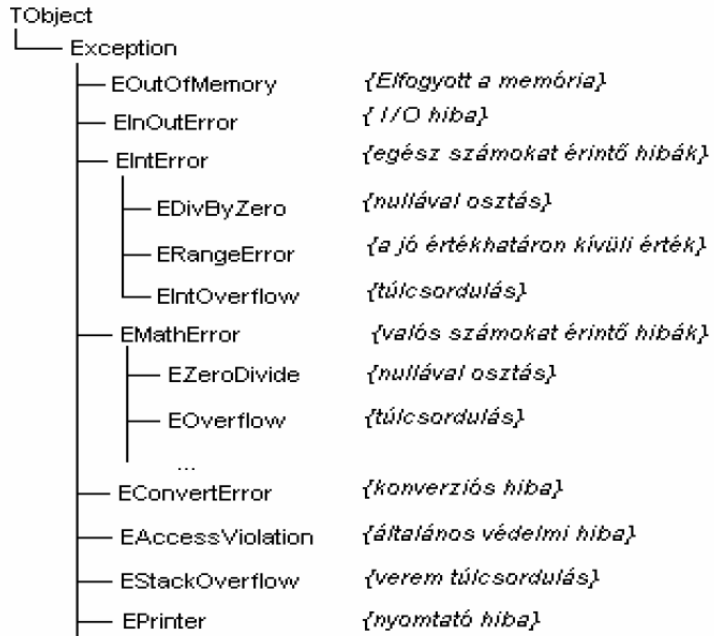
Kivételkezelés

Object Pascal komoly kivételkezeléssel rendelkezik. A hibakezelés kivételek segítségével valósul meg. A kivétel olyan hibás állapot vagy esemény, amely futásidőben keletkezik és megszakítja az alkalmazás futását, majd a futó program aktuális pontjáról a kivételkezelőhöz (exception handler) kerül a vezérlés. Az Object Pascal a kivételek megvalósítására objektumokat használ.

A kivételek objektumként történő megvalósításának előnyei:

- hierarchiába lehet rendezni jellegüktől függően, mert objektumok és érvényes rájuk az öröklődés
- egyszerűen lehet új kivételeket definiálni
- információt vihetünk a hiba keletkezési helyétől a kezelés helyére

Néhány kivételosztály a Delphi osztályhierarchiájában:



A kivételeket ugyanúgy kell deklarálni, mint egy osztályt, de célszerű ezeket az Exception osztályból származtatni.

A kivételobjektumok a SysUtils modulban definiált Exception osztályban vannak. Ha egy alkalmazás használja a SysUtils unitot, akkor minden futás idejű hiba kivétellé alakul át. A VCL teljes egészében támogatja a kivételkezelést, és a VCL-re épülő alkalmazások automatikusan használják a SysUtils unitot, ezzel engedélyezve a kivételek kezelését.

A kivételkezelés lehetőségei:

- Védelem a futás-idejű hibák ellen:

Try

utasítás blokk

Except

On *kivételosztály* **Do** *kivételkezelő blokk*

…

[Else *kivételkezelő blokk]*

End;

A kivételek lekezelésére a *try - except - end* utasítás használható. Mikor a program a végrehajtás során ehhez a részhez ér, megpróbálja elvégezni a *try* után lévő utasításokat. Ha ez sikeres volt, akkor az *end* utasítás utáni résszel folytatódik a végrehajtás.

Kivétel fellépésekor a vezérlés a legbelső *try - except* utasításhoz kerül. Az *except* utasítás utáni részben találhatóak a kivételkezelők, melyeket az *on* kulcsszó vezet be, a kivétel típusa után *do* kulcsszóval a kivételkezelő neve szerepel.

Ha a legbelső *try - except* blokk nem alkalmas az adott kivétel kezelésére, akkor eggyel kifelé lépve folytatja tovább a keresést. Ha talál olyan kivételt, amely típus kompatibilis az aktuális kivétellel, akkor a legelső ilyet hajtja végre.

Sikertelen keresés esetén az *else* ágban található utasításokat hajtja végre, ha ilyen van, ellenkező esetben a kivétel "továbbcsorog" az egy szinttel feljebb lévő blokkba, egészen addig, míg le nem kezelődik. Ha egyetlen blokkban sem kezeltük le, akkor a Delphi alapértelmezett kivételkezelője fut le, amely megjelenít egy hibaüzenetet, és ha nem végzetes hiba történt folytatja a program futását.

Ha a kivételblokkban nincsen *on - do* megadva, akkor ezt minden kivétel esetén találatnak minősíti, és végrehajtja az itt lévő utasításokat. A kivételek hierarchiájának figyelembevételével a kezelő meghívódik minden olyan esetben is, ami-

kor az általa hivatkozott kivétel egy alosztályáról van szó, tehát itt is megfigyelhető a polimorfizmus. Éppen ezért az általánosabb kezelőket az *except* blokk végére tegyük, de legalábbis minden belőlük származó kezelő után.

Kivétel csak a *raise* utasítás hatására vagy a *try* részben fellépő hiba esetén keletkezhet.

Ha a kivételkezelőben magában is kivétel lépne fel, akkor ezt még ezen belül le kell kezelni, különben az eredeti kivétel elvész, és az új kivétel lép a helyébe, ugyanis Delphiben egyszerre csak egy kivétel lehet aktív.

- Erőforrások biztonságos használata:

erőforrások lefoglalása

Try

erőforrások használata

Finally

erőforrások felszabadítása

End;

Előfordulhat, hogy bizonyos utasításokat mindenféleképpen végre kell hajtani, akkor is, ha a futás során kivétel lépett fel. Tipikusan ilyenek az erőforrás felszabadítási műveletek, így például a megnyitott file-t le kell zárni, attól függetlenül, hogy volt - e hiba a feldolgozás során. Erre szolgál a *try - finally* utasítás. Használatakor a *try* rész végrehajtása után elvégzi a *finally* részben lévő utasításokat, majd ha volt kivétel, ez automatikusan újra fellép a végrehajtás után.

Delphiben a *try* blokkot követheti egy *except* vagy egy *finally* blokk, de mindkettő nem. Erre megoldás a *try - finally* és a *try - except* utasítás egymásba ágyazott használata:

Try

file megnyitása

Try

feldolgozás

Finally

lezárás

End;

Except

hibakezelés

End;

Ekkor vigyázni kell, mert a *try-finally* -ban keletkezet *exception-t* a lezárás részben ki váltódott, lekezeletlen kivétel "felülírhatja", és így elveszítjük a kivételünket. Mindig csak egy kivétel lehet aktív.

- Kivételek kiváltása

  raise *[kivételpéldány[at címkifejezés]]*

  Egy fellépő kivételt a *raise* utasítás segítségével válthatunk ki. Használata esetén nem adja vissza a vezérlést a hiba keletkezési helyére, hanem ez átadódik a kivételkezelőnek.
  A *raise* utasítás argumentumában nem osztályt, hanem objektumot kell írni. Ezt az objektumot közvetlenül az argumentumban hozzuk létre az adott kivételosztály konstruktorának meghívásával. A kivétel kezelése után a kivételobjektum automatikusan törlődik a memóriából.

Sok beépített kivétel is található az Object Pascalban, amik általában valamilyen futásidejű problémára vonatkozna, mint például a nullával való osztás. Létrehozhatunk saját kivételeket is, amik az Exception osztály leszármazottai lesznek.

Delphi alkalmazások felépítése

A Delphi az egyes alkalmazások fejlesztése során projekteket hoz létre. Egy projekt általában a következő állományokat tartalmazza:

- Projektállomány (*.DPR): Az alkalmazás főprogramja.
  A projekt állomány szerkezete programfejből, hivatkozási részből és végrehajtható részből áll.

```
program Project1;
uses
    Forms,
    //Egységnév in 'Egységállomány'{ŰrlapNév}
    Unit1 in 'Uniti.pas' {Form1};
```

```
{$R *.RES}
begin
    Application.Initialize;
    Application.CreateForm(TForm1, Form1);
    Application.Run;    //Ez tartalmazza az üzenetkezelő ciklust,
                        //mely minden Windows-s alkalmazásban azonos
end.
```

A Programfej tartalmazza a projektállomány és a futtatható állomány nevét, amit a *program* kulcsszó után adunk meg.

A hivatkozási részben az alkalmazás által használt beépített és saját egységeinket illetve a hozzájuk tartozó űrlapállományok listáját soroljuk fel a *uses* kulcsszó után.

A {$R *.RES} egy fordítási direktíva, mely a szerkesztőnek szól, hogy a *.RES kiterjesztésű külső erőforrás állományokat szerkessze az EXE-hez.

Végrehajtó rész a *begin-end* között szerepel és feladata az alkalmazás inicializálása, futtatása, befejezése.

Az Application a TApplication osztály egy példánya, mely globális változó és az alkalmazás példányát azonosítja. A TApplication osztály metódusai az alkalmazás inicializálását, futtatását és leállítását végzik. Az osztály tulajdonságai megteremtik a Windows rendszer és az alkalmazás közötti kapcsolatot, az osztály eseményei reagálnak az alkalmazáshoz érkező üzenetekre.

- Form fájlok (*.DFM): Külalakilag leíró állomány.
  A rendszer az adott űrlap grafikus tulajdonságait tárolja bináris formában.
- Űrlaphoz tartozó egység (*.PAS): Viselkedést leíró rész.
  Adott űrlaphoz tartozó egység ezért bizonyos részei űrlap-specifikusak. A unitok zárt, önálló modulok, adott céllal.

Modulokat írhatunk mi is, de a Delphinek is vannak olyan saját belső egységei, amelyekben a Delphi eljárásait, függvényeit, objektumait stb. helyezték el. Az általunk írt unitok lehetnek formhoz kötöttek, vagy állhatnak form nélkül, önmagukban. Elsősorban nagyobb projektek esetén célszerű a logikailag, működés szempontjából együvé tartozó programrészeket egy egységbe összevonni. Így a programunk áttekinthetőbb és a fordítás is sokkal gyorsabb lesz.

Egy Delphi unit a következő fő részekből áll:

- unitfej,
- az illesztő rész *(interface),*
- a deklarációs rész (type, var stb.),
- a kifejtő rész *(implementation),*
- az egységzárás *(end.).*

```
unit Unit1;
interface
uses
        Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
        Dialogs;
type
        TForm1 = class(TForm)
private
        { Private declarations }
 public
        { Public declarations }
end;
var
        Form1: TForm1;
implementation
        {$R *.DFM}
 end.
```

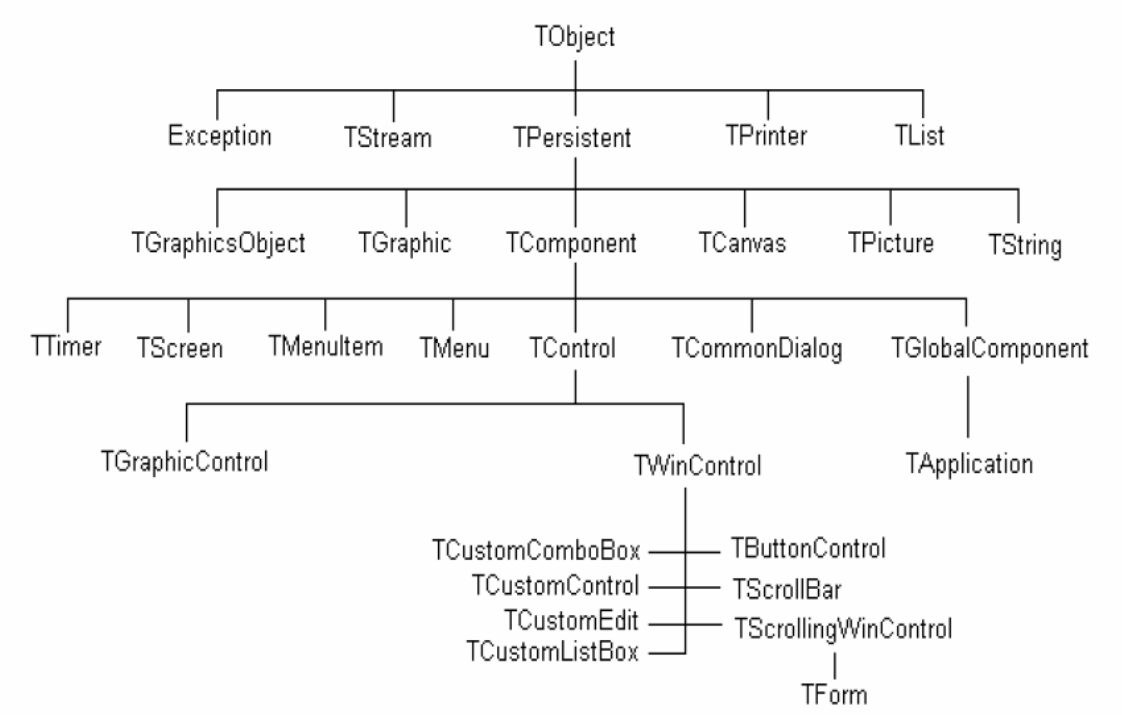
Az illesztő részben, az *interface* kulcsszó után definiálhatjuk azt, amit más programrészek is láthatnak, egyébként csak az adott unit számára elérhetők. A *uses* kulcsszó után kell felsorolni, hogy melyik más unitok tartalmát akarjuk használni.
A deklarációs részben a unit számára látható típus-, változó-, konstans- és címkedeklarációkat kell megadnunk.
Az után az *implementation* részben kell a unitot alkotó eljárásokat, függvényeket leírnunk. Ezek közül kifelé csak azok láthatók, ha az *interface* után felsoroljuk az eljárás- és függvényfejeket.
Az egységet *end.* zárja le.

Vizuális komponens könyvtár (VCL)

A Delphiben kétféle programozási stílussal kell dolgoznunk. Az egyik a komponensekkel való programozás, a másik a "hagyományos" programozási mód: a probléma megoldásának Object Pascal programnyelven való megfogalmazása.

A komponensek előre elkészített, újrafelhasználható, deklarált építőelemek, melyek egy osztályhierarchiát alkotnak és közvetlenül vagy közvetve a TComponent absztrakt osztályból származik. Ez az osztályhierarchia a Delphi működésének középpontjában áll és Vizuális Komponens Könyvtárnak (Visual Component Library), röviden VCL-nek nevezzük.

VCL osztályhierarchia:



A VCL alapját a TObject ősosztály képezi, aminek a rendszer minden osztálya a leszármazottja, így tehát a hierarchia minden tagjának egyetlen közös őse van.

A Delphi egyik legfontosabb eleme a komponens, amelynek attribútumai, metódusai, illetve eseményei lehetnek. Minden komponensnek van neve és szülő objektuma. A komponensek adott feladatra tervezett elemek, interface specifikációk, amelyeket Object Pascal-ban írtak meg. A komponenseket a nevük elé írt T betűről ismerhetjük fel és az űrlapon (formon) helyezhetjük el. A form komponens neve TForm, a ráírt szöveg TLabel, a

ráhelyezett nyomógomb TButton, stb.. Amikor az űrlapon elhelyezünk egy komponenst, a Delphi automatikusan beilleszti a *uses* kulcsszó után a megfelelő unitot.

A Delphi komponensek többségének van egy alapeseménye. Ha a fejlesztés során kétszer kattintunk a formon elhelyezett építőelemen, akkor ez az alapesemény lesz aktív, és a kódszerkesztőben ez jelenik meg.

A Delphiben való programozás tulajdonképpen a komponensek eseményein alapszik, amelyeket kiválthat a felhasználó vagy maga a rendszer. A legtöbb Delphi esemény akkor következik be, amikor egy megfelelő Windows üzenet érkezik. Ha a felhasználó kérni szeretne valamit a programtól, akkor egér vagy más beviteli eszköz segítségével hat a rendszerre, vagyis eseményt vált ki. Az eseményt a Windows rendszer fogadja, majd üzenetté alakítja, azonban az események nem egyeznek meg egy az egyben az üzenetekkel. Egy esemény egy ablaknak küldött üzenet eredménye, és ez az ablak válaszolhat az üzenetre.

A VCL hierarchiáját három fő területre oszthatjuk fel:

- komponensekre
- általános objektumokra
- kivételekre

Vannak előre definiált komponensek (ezeket láthatjuk a komponens palettán), illetve a programozó is készíthet komponenseket.

A komponensek két nagy csoportra oszthatók:

- Vezérlők (control), más néven látható (vizuális) komponensek a TControl osztály leszármazottai. Ezen osztály komponensei kitüntetett komponensek, melyek annyival tudnak többet az általános komponenseknél, hogy tervezési időben adhatóak a formhoz, illetve bizonyos esetekben a tulajdonságait is állíthatjuk, amit futásidőben megtartanak.

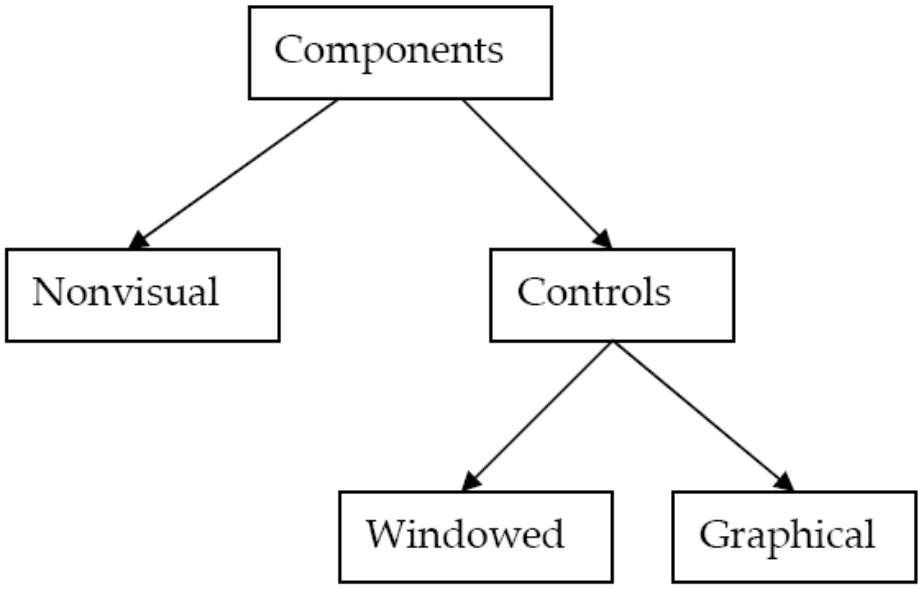
  Két típusát különböztetjük meg:

  - Ablak alakú vezérlők (window-based), melyek rendszerablakra épülnek. Ezek a komponensek a TWinControl osztály leszármazottai és ablakleíró tulajdonságokkal rendelkeznek, melyet a *Handle* tulajdonságon keresztül érhetünk el. Felhasználó szemszögéből a legfontosabb tulajdonságai közzé tartozik, hogy tartalmazhatnak más komponenseket és a beviteli fókuszt is megkaphatják. Ilyen komponens például a TEdit.

- Grafikus vezérlőknek a nem ablak alapú vizuális komponenseket nevezzük. A grafikus vezérlők sok szempontból hasonlítanak az ablakos vezérlőkre, azonban ezeknek a komponenseknek nincs ablakleírójuk, így kevesebb rendszererőforrást igényelnek. Továbbá nem tartalmazhatnak egyéb komponenseket és nem kerülhetnek fókuszba. Ilyen komponens például a TLabel

- Nem látható komponensek csak a fejlesztői környezetben jelennek meg ikonként, a lefordított alkalmazásban a háttérből végzik a tevékenységüket (pl. TTable), vagy tartozhat hozzájuk megjeleníthető felület (pl. standard dialógus ablak).
  A nem vizuális komponensek a TComponent osztályból származó osztályok, amelyek nem leszármazottai a TControl osztálynak.

Öröklődési struktúra:



Minden komponensre jellemző alapvető tulajdonságok:

- Name: a komponens egyedi neve, mellyel a komponensre hivatkozunk
- Owner: a komponens tulajdonosa. A tulajdonos feladata az, hogy megszűnésekor a tulajdonában lévő elemeket is megszűntesse
- Tag: tetszőleges egész értéket (long integer) tárolhatunk benne

Általános VCL tulajdonságok

Align	Igazítás a szülőkomponensen belül
BorderWidth	Keret vastagsága
Caption	A komponens felirata
Components	Komponens lista
ComponentCount	Komponensek száma
Color	A vezérlő elem színe
Enabled	A komponens engedélyezett, aktív-e
Font	A komponens feliratának betűformátumát tartalmazza
Height	Magasság
Hint	A kontrol alatt megjelenő útmutatás (tipp) szövege
Left	Bal betolás
Name	Egyedi neve a komponens példánynak
Owner	A komponens tulajdonos
Parent	Szülője: az az ablakozott komponens, mely a komponenst tartalmazza
ParentColor	Igazra állítva a vezérlőelem színe meg fog egyezni a szülőjével
PopupMenu	Az egér jobb gombjánaklenyomására megjelenő gyorsmenü megadása
ShowHint	Tartozzon e leíró címke a vezérlőhöz, vagyis a tipp engedélyezése/tiltása
TabOrder	A szülő kontrol által megadott tabulátor-sorrendet adja vissza
TabStop	Megadja, hogy a kontrol elérhető-e TAB billentyű megnyomásával
Top	Magasság a szülőn belül
Visible	Megadja, hogy a kontrol látható-e

Általános VCL metódusok

CanFocus	A vezérlő fókuszba kerülhet-e
Create	Konstruktor
Destroy	Destruktor, javasolt a Free használata
Focused	Megadja, hogy a vezérlől fókuszban van-e
Free	Memóriából törli
GetTextBuf	Visszaadja a vezérlő szövegét, vagy feliratát
GetTextLen	Visszaadja a vezérlő szövegének, vagy feliratának hosszát
Hide	Elrejti, láthatatlanná teszi a vezérlőt
Invalidate	Feltétel nélkül újrarajzolja a vezérlőt
ScaleBy	Átméretezi a vezérlőt az adott %-al
SetFocus	A fókuszt a vezérlőnek adja
SetTextBuf	Beállítja a vezérlő szövegét vagy feliratát
Show	Láthatóvá teszi a vezérlőt

Általános VCL események

OnChange	Változik az objektum, vagy adata
OnClick	Az egér bal gombjának kattintására következik be
OnDblClick	Az egér dupa kattintására következik be
OnEnter	Akkor következik be, ha a vezérlő megkapja a fókuszt
OnExit	Akkor következik be, ha a vezérlő elveszíti a fókuszt
OnKeyDown	Lenyomunk egy billentyűt
OnKeyPress	Leütünk egy billentyűt
OnKeyUp	Felengedjük a billentyűt
OnMouseMove	Az egérmutató a komponens felett mozog
OnResize	Az átméretezés véget ért

IV. Adatbázis-kezelés a Delphiben

BDE

A Borland Delphi napjaink egyik legfejlettebb alkalmazás-fejlesztő rendszere. Elsősorban adatbázis-rendszerek készítésére szánják, de gyakorlatilag bármilyen szoftvert el lehet készíteni vele. Ahhoz, hogy egy Delphiben írt alkalmazás meg tudjon nyitni egy helyi adatbázist, vagy csatlakozni tudjon egy adatbázisszerverhez, szüksége van a Borland Database Engine-re, röviden BDE-re, mert a Delphi adatbázis-alkalmazások nem közvetlenül kommunikálnak az adatforrásokkal, az adatbázisfájlokat nem tudják közvetlenül kezelni.

A Delphi adatbázis-kezelése teljes mértékben a BDE-re épül, a Delphi minden adatbázis-kezelő komponense ezt használja az adatok közvetett vagy közvetlen eléréséhez. A BDE egységes felületet biztosít a különböző formátumú adatbázisok eléréséhez, így nem kell ismernünk az egyes rendszerek sajátosságait.

A BDE gyakorlatilag DLL-ek és utility-k gyűjteménye, egy Windows alapú 32 bites adatbázis-motor. Feladata, hogy kapcsolatot teremtsen a fizikai adatbázis és az adatbázist kezelő alkalmazások között. Kezeli a táblaformátumokat és alapvető adatbázis-kezelő rutinokat tartalmaz. Ezért ha egy olyan alkalmazást készítünk, amely mögé adatbázis szükséges és azt telepítjük egy felhasználó gépére, ahhoz, hogy működjön a program, a BDE-t is telepíteni kell ugyanarra a gépre.

A Delphiből a következő adatbázis elérések lehetségesek:

- Natív driverekkel: a Delphi telepítése során kerülnek a gépre
  - dBASE,
  - FoxPro,
  - Paradox,
  - Access,
  - DB2,
  - InterBase,
  - Oracle,
  - Sybase,
  - MicrosoftSQL,
  - Informix.

- ODBC (Microsoft Open Database Connectivity) driverrel:
  A BDE minden olyan típusú adatbázisfájl elérését támogatja, amelyekhez létezik egy megfelelő ODBC illesztőprogram.
- Borland SQL Links-el: helyi és távoli SQL kiszolgálók eléréséhez

A programozó ezek közül választhat anélkül, hogy módosítaná programját, mert minden formátumnak saját, független meghajtója (drivere) van.
A komponensek a BDE API (Application Program Interface) függvényeken keresztül működtetik a BDE-t. A BDE függvényei objektum-orientáltán épülnek fel, így azok elérése egyszerű és strukturált. A Delphi minden verziója tartalmaz ún. STANDARD drivereket, melyekkel a lokális adatbázisok (Paradox, dBASE, InterBase SQL) érhetők el.
Ahhoz hogy különböző adatbázisszerverekhez (Oracle, Sybase, InterBase stb.) is csatlakozni tudjunk a Client/Server verzióra van szükség. Az adatbázisszerverek eléréséhez a BDE az ún. SQL-links csomagot használja, ami kiegészítő illesztőprogramokat tartalmaz a BDE-hez.
A DBE felépítése:

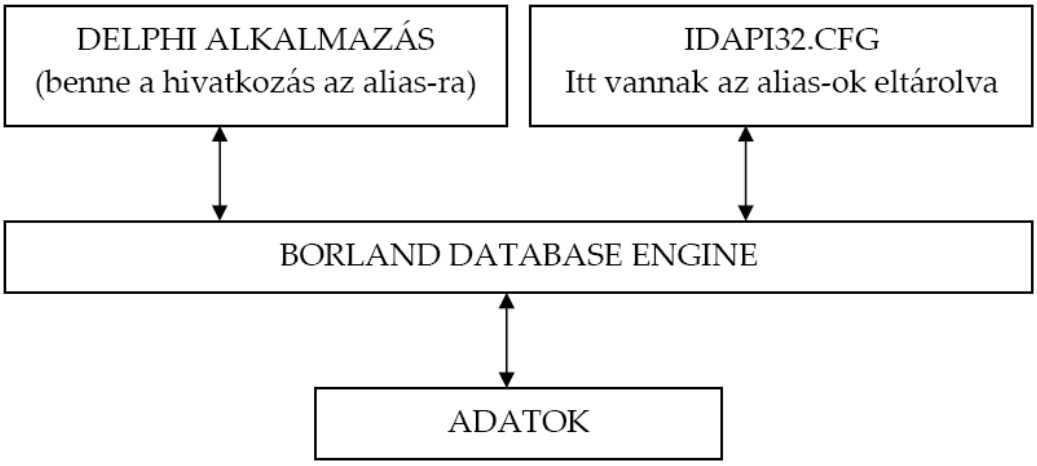
- BDE mag (BDE core): a rendszer magját képező .DLL fájlok tartoznak ide
- BDE API függvények: alapvető adatbázis-műveleteket tartalmazó függvénytár, mely adatbázis kezelésre, konfigurálásra, hibakezelésre, lekérdezésre, tranzakciókra használható függvényeket tartalmaz.
- Adatbázis meghajtók (Database Drivers): ide tartozik az öt standard meghajtó (Paradox, dBASE, FoxPro, Access, Text).
- Opcionális meghajtók (Optional Drivers): egyéb meghajtókat tartalmaz (InterBase, DB2, Informix, Oracle. Sybase, Microsoft SQL Server)
- Query motorok (Query Engines): az SQL (Structured Query Language- Strukturált Lekérdező Nyelv) motorok
- BDE Administrator: a BDE beállításait végző program
- Database Desktop: adatbázisfájlok kezelését végző segédprogram amellyel egyszerűen hozhatunk létre, kitölthetünk, vagy módosíthatunk különféle formátumú táblákat, illetve SQL lekérdezéseket is készíthetünk.

A BDE fontos szolgáltatása még az ún. alias-ok kezelése.

Alias

A Borland adatbázis motor alias-okat (álneveket) használ a különböző adatbázisokra való hivatkozáskor. Álnevek segítségével elnevezhetjük adatbázisunkat, hogy később ezen a néven hivatkozzunk rá, annak fizikai helyétől függetlenül. Az alias gyakorlatilag paraméterek halmaza, ami standard esetben, lokális adatbázisoknál egy egyszerű mutató, ami egy adott könyvtárra mutat, azaz az adatbázis elhelyezkedését (pl. c:\mydatabase) és típusát tartalmazza (dBASE, Paradox). Adatbázisszerverek esetén plusz paramétereket, egyéb információkat is tartalmaz, amelyek a szerverhez csatlakozáshoz szükségesek (pl. a szervernév, felhasználónév, jelszó, stb.).
Adatbázis-alkalmazás készítésekor a használt adatbázisra célszerű alias-sal hivatkozni, mert ha a későbbiekben, pl. megváltozik az adatok elérési útvonala, azt bármikor egyszerűen megváltoztathatjuk, mert nincs fixen belefordítva a programunkba.
A létrehozott alias a BDE saját konfigurációs állományában (IDAPI32.CFG) kerül elmentésre, és mindaddig megmarad, amíg nem töröljük.

Alias-ok működése:



Alias-ok használatának előnyei:

- nem kell újrafordítani az alkalmazást minden áthelyezés után
- a tervezéskor elég egy rövid álnevet begépelni, nem kell a teljes elérési útvonalat használni
- az alkalmazásunkat nemcsak az adatok helyétől függetleníti, hanem a formátumától is
- SQL Serverek esetén az alias sajátos információkat is tartalmazhat (pl. a felhasználó neve, stb.)

Alias-t a BDE Administrator-ral, a Database Explorer-rel, de akár saját magunk programból is létrehozhatunk.
A BDE Administrator az adatbázismotor konfigurációs programja, mellyel alias-okat hozhatunk létre, módosíthatunk, törölhetünk.
A Database Explorer egy segédprogram mellyel alias-okat kezelhetünk, adatbázisokat nézhetünk meg, módosíthatunk, SQL lekérdezéseket futtathatunk stb.

Lokális adatbázis alias-ának létrehozása BDE Administrator-ral:

- BDE Administrátor indítása
- Object/New menüpontot választása (a Databases elem legyen kiválasztva a hierarchikus felsorolásban).
- A bejövő ablakban az installált driverek közül kiválasztani azt, amelyiket használni akarunk. Legyen ez most a STANDARD (lokális dBASE vagy Paradox táblákhoz ez kell).
- OK gomb után beírhatjuk az alias nevét.
- Ezt követően a definíciós ablakban be kell állítani, hogy milyen típusú adatbázis-táblát akarunk használni (DEFAULT DRIVER), és a PATH-ban meg kell adni azt a könyvtárat, ahol az adatbázis-tábláink találhatók.
- Ezzel elkészült az új BDE álnév, zárjuk be a BDE Administrator-t, és arra a kérdésére, hogy elmentse-e a változtatásainkat válaszoljunk „igen"-nel
- Ha most elindítjuk a Delphi-t és a form-ra teszünk egy adatelérési komponenst (pl. Table), akkor annak a DatabaseName Property-ének legördülő menüjében láthatjuk a frissen létrehozott álnevünket is.

Az adatbázis-kezelést segítő segédprogramok

- BDE Administrator: adatbázis motor (Database Engine) konfigurációs programja, mellyel álneveket lehet létrehozni, módosítani, törölni és új ODBC adatforrásokat lehet hozzáadni. Az IDAPI32.CFG állományba ír, ezen az állományon keresztül kommunikál a BDE-vel.
- Database Explorer: csak 32 bites változatok Client/Server csomagjában található meg. Álnevek kezelésére, adatbázisok szerkezetének és tartalmának megtekintésé-

re, módosítására, SQL lekérdezések begépelésére és lefuttatására valamint adatszótárak kezelésére használható.

- Data Migration Wizard: különböző adatbázisok közötti metaadat (adatbázis szerkezeti információja) és adat áthelyezésére használható.
- Database Desktop: használható helyi adatállományok kezelésére, SQL nyelvű lekérdezések szerkesztésére, végrehajtására helyi, vagy távoli gépen, QBE (Query by example) formátumú lekérdezések szerkesztésére, mentésére, betöltésére, alias-ok létrehozására, törlésére.
- SQL Monitor: SQL lekérdezések nyomkövetési segédprogramja. A BDE által az SQL szerver felé küldött SQL utasítások megfigyelésére használjuk.
- Server Manager: az InterBase adatbázis-szerver karbantartó és felügyeleti programja, mely alkalmas a felhasználók kezelésére, adatbázisok mentésére.
- Windows ISQL: az InterBase adatbázisok SQL parancsok általi közvetlen kezelésére alkalmas.

A Delphi adatbázis-kezelési komponensei

Az adatbázisokhoz kapcsolódó komponensek két csoportba oszthatóak:

- Adatelérési komponensekre: melyek a Data Access palettán található nem vizuális komponensek, azaz csak tervezési időben, ikonként láthatók és gyakorlatilag ők kommunikálnak a BDE-vel.

- Adatmegjelenítési komponensekre: melyek a Data Controls palettán található vizuális komponensek, és a segítségükkel az adatokat láthatóvá és szerkeszthetővé tehetjük. Az inputot és az outputot valósítják meg.

Adatelérési (Data Access) komponensek

Ide azok a komponensek tartoznak, melyek segítségével megteremthetjük az adatbázis fájljaival való kapcsolatot, hozzáférhetünk az adatbázisban tárolt adatokhoz, módosíthatjuk azokat, illetve lekérdezéseket készíthetünk.
Ezek a komponensek nem vizuális komponensek, csak tervezési időben, ikonként látható a fejlesztői környezetben és gyakorlatilag ők kommunikálnak a BDE-vel.

TDataSource:

Célja egy tábla, lekérdezés adatainak elérhetővé tétele, de nem kapcsolódik közvetlenül a fizikai táblához. Az adatforrás kapcsolatot teremt az adathalmaz (DataSet) komponenseivel és az adatmegjelenítő (Data Controls) komponensek között.
A DataSet komponensek azok a komponensek, amelyek valamilyen adathalmazt biztosítanak. Lehetővé teszi többrétegű alkalmazások készítését is.
Fontosabb tulajdonságai:

- AutoEdit: az adatkészlet szerkeszthetővé tétele, ha egy adatfüggő komponens fókuszt kap
- DataSet: mely komponenssel (Table, Query) áll kapcsolatban
- State: adatkészlet állapota (dsBrowse, dsEdit, dsInsert, dsInactive, stb.)

TTable:

A TTable komponens a BDE-t használva hozzáférést biztosít egy adatbázis-táblához, és kezelhetővé válnak az adatok. A TDataSet osztály leszármazottai, paramétereit rendszerint tervezési időben határozzuk meg. Használatakor meg kell adni a használt adatbázis nevét a DataBaseName tulajdonságánál, ami lehet az adatbázis álneve, vagy elérési útvonala, illetve a tábla nevét a TableName tulajdonságnál.
A TTable eseményei között megtaláljuk a navigáció és az adathozzáférés összes eseményét, ezen kívül hibakezelést is végezhetünk, metódusaival adatbázis-műveleteket végezhetünk.
Fontosabb tulajdondágai:

- ReadOnly: igazra állítva meggátolja az adatok megváltoztatását
- Exclusive: megtiltja, hogy egyszerre több folyamat használja a táblát
- Indexekre vonatkozó tulajdonságok
- Active: megnyitás, bezárás egy módja

TQuery:

Ennek a komponensnek a segítségével olyan DataSet-et építhetünk be a programunkba, ami SQL kifejezésen alapul. A TQuery komponenssel különféle SQL utasításokat adhatunk meg, és lekérdezéseket futtathatunk.

Hasznos komponens, mert ezzel egyszerre több táblát is elérhetünk, összekapcsolhatunk, illetve számításokat, rendezéseket és leválogatásokat végezhetünk az SQL nyelvből adódóan. A Query által visszaadott adatok nem módosíthatók, de magával a Query-vel végezhetők adatmódosító műveletek.

TStoredProc:

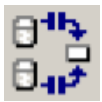
Szerver-kliens adatbázis-szerkezetnél használatos, amikor a kliens az adatbázis-szerveren eljárásokat szeretne tárolni, illetve azokhoz hozzáférni. A tárolt eljárás egy SQL szerver lokális eljárására mutat, ami az adatbázis-kezeléshez kapcsolódó gyakran használt módszer. Lefuttatása után az eredményeket tábla formájában felhasználhatjuk

TDatabase:

A TDatabase komponens egy konkrét adatbázis elérését biztosítja. Paradox táblák esetén nem kell külön Database objektumot használni, mert a TTable és TQuery komponensek DatabaseName jellemzőjükkel nem egy adatbázis-komponensre hivatkoznak, hanem egy alias-ra.

Rendszerint szerver-kliens architektúráknál használatos, mert ez biztosítja az utat az adatbázis felé, az adatbázis-kapcsolat magas szintű vezérelhetőségét teszi lehetővé.

Csatlakozáskor felelős a belépési folyamat ellenőrzéséért, metódusaival tranzakciókezelést tudunk lebonyolítani.



TSession:

Minden adatbázisos alkalmazás indításakor automatikusan létrejön egy Session objektum.

A TSession komponens feladata az adatbázis-alkalmazás kapcsolatainak felügyelete. Jellemzői, metódusai nem egy adott adatbázisra, hanem az egész munkafolyamatra vonatkozik.

A Session objektum segítségével létrehozatunk alias-okat az AddStandardAlias és az AddAlias metódusokkal, de ahhoz, hogy a BDE az új alias-t el is mentse, meg kell hívni a SaveConfigFile() metódust.
Létező alias-okat törölhetünk a DeleteAlias metódussal.
Az automatikusan létrejövő Session komponensen kívül szükség van további TSession komonensekre, ha:

- az alkalmazásunk különböző hálózati gépeken található Paradox táblákat kezel, mert egy Session komponens csak egy hálózati gép adatbázisait képes ellátni.
- az alkalmazásunk több szálon egy közös adatbázist kezel, mert minden szál egy-egy munkafolyamat és minden munkafolyamatnak saját Session komponenssel kell rendelkeznie.

TBatchMove:

A TBatchMove komponens segítségével, mint ahogy a nevéből is látszik, kötegelt műveletek végrehajtásra nyílik lehetőségünk. Ilyen művelet lehet rekordok másolása, mozgatása, vagy törlése egy adathalmazból, de fontos, hogy forrásnak és célnak TTable típust kell megadni.

TUpdateSQL:

A TUpdateSQL komponens kifejezetten adatmódosításra szolgál. Segítségével adatfrissítő műveleteket végezhetünk külön-külön SQL lekérdezések segítségével.
Rendszerint a táblák és a lekérdezések UpdateObject értékeként használjuk.

Adatmegjelenítési (Data Control) komponensek

Adathalmazok (DataSet) megjelenítését a felhasználói felületen az adatmegjelenítő komponensek végzik.
Az adatelérési komponensek csoportjába tartoznak mindazon objektumok, amelyek segítségével tetszőleges típusú, elérésű és formátumú adathalmazok tartalmához férhetünk hozzá, jeleníthetjük meg, vagy szerkeszthetjük. Ide sorolhatjuk az alkalmazás és adatbáziskezelő kommunikációs felületének kezelésére szolgáló objektumokat, a fizikai adatbázist, az adatbázisban tárolt adattáblákat, az SQL alapú lekérdezéseket, a tárolt eljárásokat reprezentáló komponenseket.

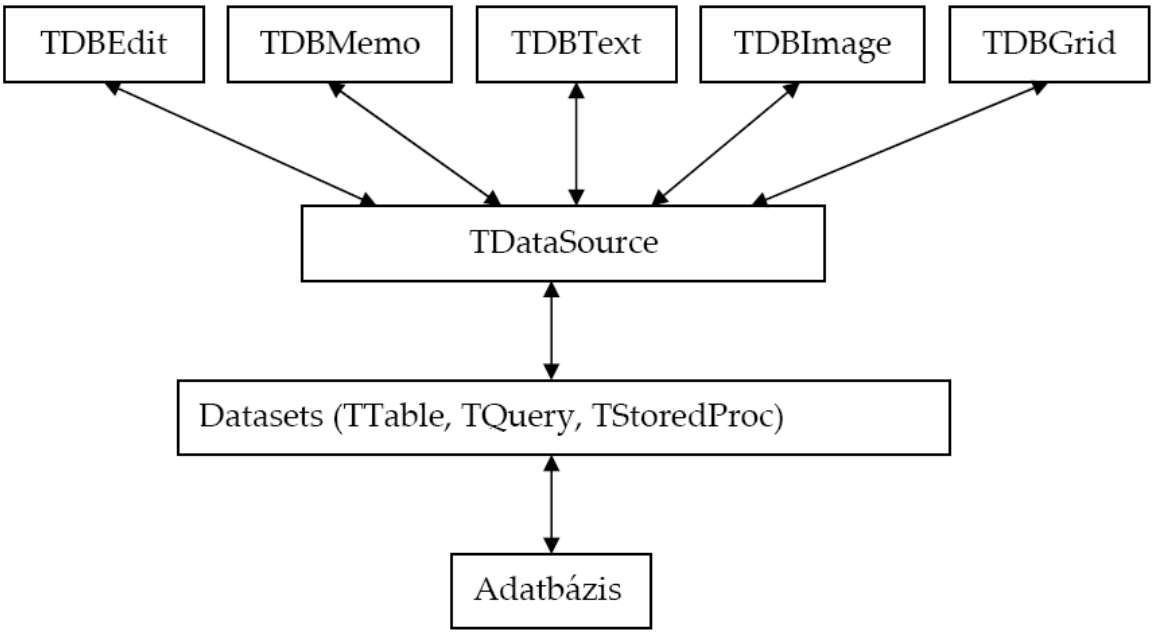
Az adatelérési (Data Access) és adatmegjelenítési (Data Control) komponensek nem tudnak közvetlenül egymáshoz kapcsolódni, ezért szükség van a DataSource komponensre, melyet a TDataSource komponensosztály objektumai reprezentálnak. Először a DataSource kapcsolódik egy adatforrás komponenshez, majd ezután egy adatmegjelenítési komponens DataSource Property-ét tudjuk ráállítani az adatforrásra. Az adatmegjelenítő komponensek mindig az adatforrásra hivatkoznak, ezáltal biztosítva a programunk függetlenségét az adatbázis típusától, formátumától.

Az adathalmazok és adatforrások összekapcsolására a TDataSource komponensosztály DataSet tulajdonsága szolgál. Az adatforráshoz egyidejűleg csak egy adathalmazt lehet kapcsolni, de egy adatforráshoz tetszőleges számú adatmegjelenítési komponens kapcsolódhat.

Az adatmegjelenítési komponensek az adatokat nem a felhasználói felületről, vagy az alkalmazás kódjából kapják, hanem egy adathalmazból olvassák ki.

Mivel a komponenseknek nincs közvetlen kapcsolatuk a fizikai adathalmazzal, ezért a megjelenítés szempontjából teljesen mindegy, hogy milyen adatszolgáltató komponenstől kapják az adatokat. Ezzel a megoldással az adatbázis-kezelő programjainkban teljesen különválaszthatjuk az adatfeldolgozási, illetve adatmegjelenítő modulokat

Egy adat megjelenítése a képernyőn adatmegjelenítési komponens segítségével:



TDBGrid:

Adathalmaz rekordjainak táblázatos formában történő megjelenítésére szol-

gál. Számos property-je közül a DataSource a legfontosabb, ezt kell arra az adatforrás komponensre állítani, amely által szolgáltatott adatokat meg kívánjuk jeleníteni.

A DBGrid komponens fontos tulajdonsága az Options. Itt az egész táblázatra vonatkozó beállításokat végezhetünk.

A következő tulajdonságokat állíthatjuk be:

- dgEditing: Ha false nem szerkeszthetők az adatok a DbGridben.
- dgAlwaysShowEditor: Ha true, a grid állandóan szerkesztési módban van, így nem kell az Enter-t vagy az F2-t leütni szerkesztés előtt.
- dgTitles: Beállíthatjuk, hogy megjelenjenek-e a mezőnevek a táblázat tetején vagy nem.
- dglndicator: Megjelenjen-e a legelső oszlop előtti keskeny sáv, amiben egy kis kurzor található.
- dgColumnResize: Át lehessen-e méretezni és mozgatni az oszlopokat.
- dgColLins: Azt állítja be, hogy a sorok között legyenek-e vonalak.
- dgRowLines: Azt állítja be, hogy az oszlopok között legyenek-e vonalak
- dgTabs: Ha true akkor Tab billentyű hatására a következő oszlopba lép, egyébként a következő kontrolra az űrlapon
- dgRowSelect: Ha true, csak teljes sor jelölhető ki.
- dgConfirmDelete: Ha true, törlés előtt megjelenik egy dialógusablak, melyben meg kell erősítenünk, hogy tényleg törölni akarjuk az adott rekordot.
- dgCancelOnExit: Új rekord felvitelekor, ha nem viszünk fel egyik mezőbe sem adatot, akkor nem veszi fel az üres rekordot a táblába.
- dgMultiSelect: Ha true, akkor egyszerre több sor is kijelölhető

A komponens másik fontos tulajdonsága a Columns, mellyel mezőket vehetünk fel a gridbe, törölhetünk onnan, megváltoztathatjuk a mezők sorrendjét az adathalmazban lévő sorrendtől eltérőre, beállíthatjuk, hogy az adott mezőben az értékek merre legyenek igazítva, beállíthatjuk a mező megnevezését, stb. Mindezt az oszlopszerkesztőbe, a Colums Editor-ban végezhetjük el.

Egy mezőobjektum fontosabb tulajdonságai:

- Alignment: Az adatok igazítása a mezőben
- ButtonStyle: Ha szerkeszthető a cella, akkor beállíthatjuk az adott cella tulajdonsá-

gát úgy, hogy egy Elipsis Button jelenjen meg benne, mellyel pl. egy másik beviteli űrlapot jelenítünk meg, vagy egy legördülő menü jelenjen meg, melyből előre definiált értékek közül választhatunk, hogy mit akarunk felvinni a cellába, illetve az adott rekord adott mezőjébe.

- Color: A mező színe.
- PickList: Ha a ButtonStyle=cbsAuto, akkor az ide felvitt elemek megjelennek egy legördülő menüben az adott mező egy cellájának szerkesztésekor.
- ReadOnly: Ha true, a mező csak olvasható
- Title: A mező fejlécének tulajdonságainak beállítására szolgál.
- Visible: Ha false, a mező nem látható.
- Width: A mező szélessége.

TDBNavigator:

Navigációs gombokat tartalmazó vizuális komponens, mely segítségével a felhasználó a rekordmutató léptetésén túl, felvehet új rekordot, törölheti az aktuális sort, vagy akár szerkesztheti is azt. Néhány fontosabb tulajdonság:

- DataSource: az adathalmazra kell állítani.
- Hints: segítségével egy string listában minden egyes gombra, beállíthatjuk, hogy mi jelenjen meg útmutatásként, ha az egeret az adott gomb fölé visszük.
- Showhint: true-ra kell állítani, ha azt akarjuk, hogy a hint-ek megjelenjenek.
- ConfirmDelete: ha igaz, akkor a törlés gombot megnyomva egy dialógusablak jelenik meg, ahol meg kell erősíteni a törlési szándékot.
- VisibleButtons: el tudjuk tüntetni azokat a gombokat, amelyekre nincs szükségünk az adott feladathoz.

TDBText:

A DBText statikus adatmegjelenítő komponens. Akkor használjuk, ha a formon nem módosítható szöveget kell megjeleníteni egy relációs táblából. Az éppen aktuális rekord hozzárendelt mezőjének az értékét jeleníti meg.

A Datasource és a DataField tulajdonságokat kell beállítani ahhoz, hogy megjelenjen az adat a komponensben. Ezeket a tulajdonságokat leszámítva, nem rendelkezik speciális tulajdonságokkal.

TDBEdit:

Az egyik leggyakrabban használt vezérlőelem a beviteli mező adatbázisokhoz illesztett változata, a DBEdit. Nem csak a mező értékét képes megjeleníteni, hanem segítségével egy rekord mezőjét szerkeszthetjük, módosíthatjuk is, amennyiben engedélyeztük ezt az adatbázis műveletet.

A Datasource és a DataField tulajdonságokat kell beállítani ahhoz, hogy megjelenjen az adat a komponensben. Ezeket a tulajdonságokat leszámítva, nem rendelkezik speciális tulajdonságokkal.

TDBMemo:

Számos esetben előfordulhat, hogy nem elegendő egy soros beviteli mező, ilyen esetekben használhatjuk a DBMemo komponenst, amely a DBEdit többsoros változata és hosszabb szövegek megjelenítésére, szerkesztésére szolgál.

TDBImage:

A DBImage segítségével BLOB (Binary Large Object) típusú mezőben tárolt képeket tudunk megjeleníteni. Néhány tulajdonság:

> Center: a kép a DBImage közepén helyezkedjen-e el
> Stretch: a DBImage méretének megfelelően kihúzhatjuk a képet, ha kisebb nála

TDBListBox:

A DBListBox egy listaelem, amely lehetséges értékeit nekünk kell feltölteni tervezési- vagy futási időben, azonban nem jelenik meg a mező aktuális értéke. Akkor használjuk, ha egy mezőnek csak adott értékei lehetnek, hogy a felhasználó ne tudjon más értéket beállítani.

Néhány tulajdonság:

> DataField: a listából kiválasztott elem az itt beállított mezőben kerül tárolásra.
> Items: a listából választható elemeket itt adjuk meg tervezési- vagy futási időben

TDBComboBox:

A DBComboBox komponenst használva egy legördülő listából választhatjuk ki azt az elemet, amivel fel akarjuk tölteni az aktuális rekord megfelelő mezőjét.

A komponens mindig a mező aktuális értékét mutatja, és a legördülő listából választhatjuk ki az új értéket, amivel le akarjuk cserélni az aktuálisat, illetve új rekord esetén innen tudjuk felvinni a kívánt elemet. Néhány tulajdonsága:

> DataField: a szerkesztendő mezőt itt tudjuk beállítani.
> Items: a választható elemek listáját itt kell megadnunk.

TDBCheckBox:

Abban az esetben, ha egy mező értéke csak igaz, vagy hamis lehet, akkor használhatjuk a DBCheckBox vezérlőelemet. A logikai érték kerül eltárolásra az adattábla megfelelő mezőjében. Néhány tulajdonság:

> DataField: a beállított mezőt itt tudjuk beállítani.
> ValueChecked: itt adjuk meg azt az értéket, ami a DBCheckBox kijelölt állapotát jelenti.

  Pl. ha DBCheckBox1.ValueChecked:='I', új rekordot viszünk fel és a DBCheckBox-ot bejelöljük, akkor a beállított mezőbe (property) egy 'I' betű fog belekerülni. Ha tallózunk egy adathalmaz rekordjai között, akkor abban az esetben fog a DBCheckBox-unk kijelölté válni, ha az adott rekord megfelelő mezőjében T betű van, minden más esetben üres marad.

> ValueUnChecked: megadhatjuk, hogy mi jelentse a nem kijelölt állapotot.

Abban az esetben, ha mindkét előbb említett tulajdonságot beállítottuk, és az aktuális rekord adott mezője sem a ValueChecked-ben sem a ValueUnCheckedben beállított értéket nem tartalmazza, akkor határozatlan állapotban van és DBCheckbox beszürkítve jelenik meg.

Lehetőség van arra is, hogy több érték is egy állapotot jelentsen, ezt a következőképpen lehet megadni:

```
DBCheckBox1.ValueChecked := 'On;Yes;True';
```

TDBRadioGroup

A DBRadioGroup komponens egy adatbázishoz kapcsolódó választógombcsoportot hoz létre. A gombok egy listában tárolódnak és a kiválasztott elemnek megfelelő értéket lehet felvinni az adott tábla mezőjébe.

Külön állíthatjuk be a képernyőn megjelenő elemeket (Items property) és azt az értéket, amit felvisz egy táblába (Values property) ha az adott elemet kiválasztjuk. A mezőt itt is a DataField tulajdonsággal állíthatjuk be.

TDBLookupListBox

Ha az adathalmaz egy mezőjének lista alapján történő módosítását teszi lehetővé, de a lista elemeit egy másik adathalmaz mezőjéhez tartozó értékek határozzák meg. Rendkívül jól használható több táblás adatbázisok esetében a hivatkozási integritásokat tartalmazó adattáblák mezőinek szerkesztésére.

Néhány tulajdonság:

- DataSource: annak a táblának az adatforrása, aminek egy mezőjét fel akarjuk tölteni egy másik táblából származó értékkel
- DataField: annak a mezőnek a neve, amit fel fogunk tölteni.
- ListSource: annak a táblának az adatforrásának a neve, amelyből az adatot vesszük.
- ListField: Itt állíthatjuk be, hogy mely mezők jelenjenek meg a listában. Ha több mezőt is meg szeretnénk jeleníteni, akkor azokat ';'-vel kell elválasztani.
- KeyField: Annak a mezőnek a neve, aminek értékét be fogjuk írni a kiválasztott rekordból.

TDBLookupComboBox

Funkciója megegyezik a TDBLookupListBox komponenssel, de itt az adatokat legördülő listában jelenítjük meg.

TDBRichEdit

A DBRichEdit komponens nagyon hasonlít a DBMemo vezérlőelemhez, mivel itt is több soros információt tárolhatunk, azonban az ebben megjelenített tartalomhoz formátumot is hozzárendelhetünk. Olyan esetekben célszerű használni, amikor szükség van a szövegek formázott tárolására. Lehetőség van szövegek stílusát, betűtípusát, betűméretét megváltoztatni.

TDBCtrlGrid

Olyan táblázat, amely egy adatforrás rekordjait jeleníti meg, szabad formájú

elrendezésben. Ez azt jelenti, hogy a táblázat első sorában tetszőleges adatelérési komponenseket helyezünk el, amiket a program futásakor megsokszoroz, és minden egyes sorban ugyan azzal a kiosztással megjeleníti a rekordok adatait.

Néhány tulajdonság:

> DataSource: tartalmazza azt az adatforrást, amely által szolgáltatottadatokat meg akarjuk jeleníteni
> PanelHeight: egy sor hosszát határozhatjuk meg
> PanelWidth: egy sor szélességét határozhatjuk meg
> ColCount: a sorok számát tudjuk beállítani
> RowCount: az oszlopok számát tudjuk beállítani

TDBChart

Adatbázison alapuló grafikon készítésére használható komponens.

TDataModule osztály

A TComponent leszármazottja. Szerepe, hogy az alkalmazás futási idejében láthatatlan komponenseinek gyűjteményét tartalmazza. Tervezéskor egy vizuális konténerként működik, futási időben viszont az adatmodul láthatatlan. Használatának előnyei:

- Az alkalmazás űrlapjai megosztva használhatják nemcsak futási, de tervezési időben is a ráhelyezett komponenseket.
- Adatbázisos alkalmazások esetén megvalósítja az adatelérési logika és a felhasználói felület elkülönítését.
- Különböző metóduokat implementálhatunk benne, amik hívhatók lesznek az egész alkalmazásban.
- Az elkészített adatmodul kimenthető, így más alkalmazások számára is elérhető lesz.

Két leggyakrabban használt eseménye van. Az egyik az OnCreate, ami az adatmodulon található komponensek inicializálására szolgál (tábla és query komponensek megnyitása), a másik az OnDestroy, amiben végső tevékenységeket építhetünk be (tábla és query komponensek bezárása).

SQL

Az SQL (Structured Query Language) az adatbázisok kezelésére kidolgozott strukturált lekérdező nyelv. Az adatbázis-kezelő rendszerek szabványosított nyelve, mely relációs adatmodelleket kezel.

A Delphi alkalmazásokban is van lehetőség SQL utasítások futtatására a TQuery komponens segítségével. Az SQL utasítások három csoportba sorolhatók:

- DDL (Data Definition Language): Adatdefiníciós nyelv

  Ide tartoznak a táblák, indexek létrehozása, módosítása, törlése, melyekkel az adatbázis szerkezetét módosíthatjuk

  Create Table utasítással új, üres táblát hozhatunk létre zárójelek között felsorolva a tábla mezőneveit és a mezők típusait.

  Alter Table utasítással egy létező táblát tudunk átstrukturálni és Drop Table -el táblát törölhetünk.

- DML (Data Manipulation Language): Adatmanipuláló nyelv

  Olyan utasítások tartoznak ide, melyekkel az adatbázis adatait kezelhetjük. Insert-tel új adatot adhatunk a táblához, Select-tel, Update-tel és Delete-tel létező adatokat kérdezhetünk le, módosíthatunk és törölhetünk a táblából.

- DCL (Data Control Language): Adatvezérlő nyelv

  Segítségével adatbázis-kezelésével kapcsolatos feladatokat láthatunk el. Adatvédelmet szabályozó utasítások a GRANT, REVOKE; tranzakció lezárására a COMMIT, ROLLBACK parancsok szolgálnak.

V. Beszámolók, jelentések készítése a Rave segítségével

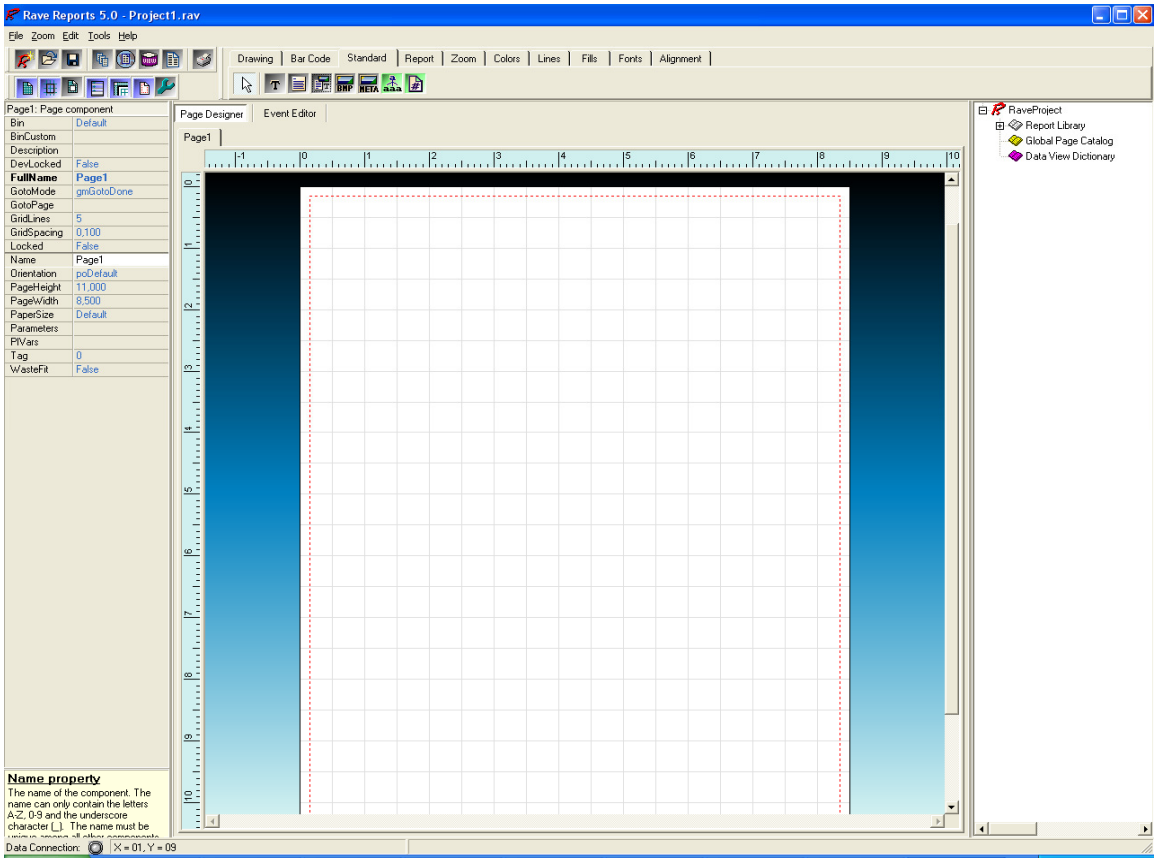
Az adatbázisprogramok lehetőséget adnak az adatok áttekintésére és módosítására, a programok kimenetét azonban gyakran papírra kell nyomtatnunk. A Delphi sokféle módon támogatja a nyomtatást a közvetlen szövegkimenettől kezdve a Canvas használatáig, a kifinomult adatházis-jelentésektől kezdve a különféle formátumú (Microsoft Word, Sun OpenOffice stb.) dokumentumok előállításáig. Szakdolgozatomban a jelentések elkészítéséhez a Delphi 7-hez mellékelt, külső fejlesztőktől származó jelentéskészítő Rave motor alkalmaztam.

A jelentéskészítő eszközök igen fontosak, mivel összetett feladatokat oldanak meg: a jelentéskészítő alrendszer önálló alkalmazássá válhat.
A jelentéseket nagy odafigyeléssel kell elkészíteni, mivel a felhasználó gyakran csak ezen a felületen keresztül érintkezik a programmal, a jelentés így sokszor nagyobb jelentőséggel bír, mint a szoftver maga. A jelentést a program felhasználóján kívül valószínűleg többen is megnézik majd, ezért ügyelnünk kell a jelentések jó minőségére.

A jelentések a program által kezelt adatok bemutatásának elsődleges eszközei. A hagyományos jelentéskészítő alkalmazások az adatok áttekinthető, könnyen értelmezhető megjelenítésének problémáját sávos elrendezéssel oldották meg, amely a táblázatos megjelenítéshez állt közelebb. Manapság azonban ennél sokkal összetettebb követelmények vannak, amelyeknek az egyszerű sávos megjelenítéssel már nem tudunk megfelelni. A Rave Reports egy jelentéstervező környezet, amely számos egyedi szolgáltatással gondoskodik arról, hogy a jelentések elkészítése egyszerűbb, gyorsabb és hatékonyabb legyen. A Rave számos jelentésformátumot kezel, és a tükrözéshez hasonló haladó szolgáltatások révén megkönnyíti a jelentések módosítását, karbantartását.

A Rave: vizuális jelentéskészítő környezet

A Rave elindításához kattinthatunk a formon elhelyezett TRvProject komponensre, vagy választhatjuk a Delphi fejlesztőkörnyezetében a Tools, Rave Designer menüpontot. A Rave Designer több részből áll Page Designer (Laptervező), Event Editor (Eseményszerkesztő), Property (Tulajdonság) tábla, Project Tree (Projektfa) tábla, eszköztárak, eszköztárpaletta és állapotsor.



A laptervező és az eseményszerkesztő

A Rave Designer ablakának központi része a jelentés formáját megadó laptervező (Page Designer) és az eseményszerkesztő (Event Editor), melynek segítségével futásidőben módosíthatjuk jelentéseinket.

A laptervező a Rave egyik legfigyelemreméltóbb része. Ez a lap a jelentés alapja, ahol az összes tervezési lépést végrehajtjuk. A laptervezőben kezdetben csak egy rácsot látunk. A tervezés alatt álló lapok neveit a laptervező feletti fülekről olvashatjuk le.

Az eseményszerkesztővel egyedi programokat állíthatunk össze a jelentéskészítő komponens számára. Minden komponens rendelkezik olyan eseményekkel, amelyeket számításokhoz karakterlánc-műveletekhez vagy egyedi jelentésszerkezet összeállításához használhatunk.

A tulajdonságtábla

A Rave Designer bal oldalán látható Property (Tulajdonság) táblával átalakíthatjuk a komponensek kinézetét és viselkedését. A tábla szerepe a Delphi objektumvizsgálójához hasonló. Amikor kijelölünk egy komponenst a lapon, a tulajdonságtábla mindig a kijelölt

komponens tulajdonságait tükrözi. Ha nincs kijelölt komponens, a tábla üres marad. A Delphi fejlesztőkörnyezetéhez hasonlóan a tulajdonságok értékét itt is a mezők tartalmának szerkesztésével, a lenyíló listák valamelyik elemének kiválasztásával, vagy egy szerkesztőablakon keresztül módosíthatjuk. Azoknál a tulajdonságoknál, ahol rögzített értékkészletből választhatunk, az értékre történő kattintással elérhetjük a következő értéket.

A projektfa

A projektfa (Project Tree) a tervező Jobb oldalán helyezkedik el, és igen beszédes. A fa egyszerű hozzáférést biztosít a Jelentés felépítésében részt vevő elemekhez. A projektfa három fő csomópontot tartalmaz: Report Library (Jelentéskönyvtár), Global Page Catalog (Globális lapgyűjtemény) és Data View Dictionary (Adatnézet szótár).

Report Library Ez tartalmazza a projektben lévő jelentéseket. A jelentések egy vagy több lapból állnak, a lapok pedig egy vagy több komponenst tartalmaznak.

Global Page Catalog A jelentéshez használt sablonok kezelésére szolgál. A sablonok egy vagy több komponenst tartalmaznak, és a Rave tükrözés nevű megoldásával újrahasznosíthatjuk azokat. A sablonok tartalmazhatnak fejléceket és lábléceket, előrenyomtatott űrlapokat, vízjeleket, de akár teljes lapmeghatározásokat is, amelyeket azután újabb jelentések alapjaként használhatunk.

Data View Dictionary Itt találjuk meg a jelentésekhez használt összes adatnézetet és egyéb, az adatokhoz kapcsolódó objektumokat.

Az eszköztárak és az eszköztárpaletta.

A Delphihez hasonlóan a Rave is kétféle eszköztárat tartalmaz: komponens-eszköztárakat és a fejlesztőkörnyezet eszköztárait.

A Rave alapértelmezés szerint a következő komponens-eszköztárakat tartalmazza: Standard (Szabványos), Drawing (Rajzolás), Report (Jelentés) és Bar Code (Vonalkód).

A szerkesztő eszköztárai lehetőséget adnak a projekt és a meglévő komponensek módosítására.

Project eszköztár Hasonló a komponens-eszköztárhoz, mivel új jelentéseket, lapokat és komponenseket készíthetünk a segítségével. A Project eszköztárban Új projekteket is elindíthatunk, menthetjük és betölthetjük létező projektjeinket, az aktuális jelentést pedig a nyomtatóra küldhetjük.

Alignment eszköztár Ez az eszköztár számos olyan eszközt tartalmaz, amelyekkel a lapon elhelyezett komponensek helyzetét módosíthatjuk. A program általában az elsőként kijelölt komponenst veszi az igazítás alapjául. A komponensek nyomtatási sorrendjét a Move Forward (Előbbre), Move Behind (Hátrább), Bring To Front (Felülre) és Send To Back (Hátra) gombokkal szabályozhatjuk. A csapokkal finomabb mozgatásokat hajthatunk végre.

Fonts eszköztár Itt módosíthatjuk a betűtípusok nevét, méretét, stílusát és igazítását.

Fills eszköztár A téglalapokhoz és a körökhöz hasonló alakzatok kitöltését szabályozza.

Lines eszköztár A keretek szélességét és a vonalak stílusát módosíthatjuk itt.

Colors eszköztár Az elsődleges és a másodlagos színek (általában az előtér és a háttér színe) meghatározására szolgál. A bal egérgomb az elsődleges, a jobb egérgomb pedig a másodlagos színt jelöli ki.

Zoom eszköztár Számos olyan eszközt találunk itt, amelyekkel nagyíthatjuk, illetve kicsinyíthetjük a laptervező tartalmát. *Designer eszköztár* A laptervező és a Rave tervező testreszabására szolgál.

Az állapotsor

Az állapotsor a tervező alsó részén helyezkedik el, és a közvetlen adatkapcsolat állapotáról, az egérmutató helyzetéről és a méretezésről ad tájékoztatást. Az adatkapcsolat lámpájának színe a Rave adatrendszer (DirectDataViews) állapotáról árulkodik: a szürke az inaktív, a zöld az aktív állapotot jelzi, a Sárga és a piros pedig különféle adatelérési helyzetekre (várakozás a válaszra, időtúllépés) hívják fel a figyelmet.

Az RvProject komponens használata

A Rave jelentés egy .rav fájlban helyezkedik el. A jelentést a Rave lap segítségével kapcsolhatjuk össze Delphi 7 programunkkal. Ez a lap számos komponenst tartalmaz, ezek közül pedig az RvProject a legfontosabb. A komponens formon való elhelyezése után állítsuk be a ProjectFile tulajdonságban a megfelelő Rave fájl nevét, majd rendeljük hozzá az alábbi kódot valamelyik gomb eseménykezelőjéhez:

```
RvProject1.Execute;
```

A hivatkozott fájl teljesen elválik az alkalmazástól, és a Delphi programtól függetlenül módosíthatjuk. Ha be szeretnénk olvasztani a .rav fájlt a Delphi alkalmazásba, töltsük be azt a DFM fájlba. Ehhez a Rave projekt komponens StoreRAV tulajdonságát használhatjuk.

Adatkapcsolatok

A Delphi alkalmazásokban megjelenő és a Rave Designer-ben látható DirectDataViews elemek adatainak összekapcsolását adatkapcsolati komponensek biztosítják. A Rave jelentés és az adatkapcsolati komponens kapcsolatát az utóbbi Name tulajdonságában megadott érték határozza meg.
Az alábbi adatkapcsolati komponensekkel dolgozhatunk:

- *RvCustomConnection* Programozott események felhasználásával biztosítja az adatokat a Rave jelentés számára, és az adatbázistól független adatokat továbbít a vizuális jelentésekhez.
- *RvDataSetConnection* A TDataSeh osztály leszármazottait kapcsolja össze a Rave DirectDabaView komponensével. A FieldAliasList tulajdonságon keresztül megváltoztathatjuk az adathalmaz-mezők neveit.
- *RvTableConnection* és *RvQueryConnection* BDE táblát vagy lekérdezést kapcsol össze a Rave DirectDataView komponensével.

A Jelentés megtervezése során vizuális komponenseket helyezhetünk el közvetlenül a lapon, illetve más tároló elemek (ilyen például a Band. és a Region) belsejében. Ezek között a komponensek között van olyan is, amelyik nem kapcsolódik az adatbázis adataihoz; ilyenek például a tervező Standard eszköztárában lévő Text, Memo és Bitmap komponensek. Vannak olyan komponensek is, amelyek az adatbázistáblák mezőihez kapcsolódnak (a Report eszköztáron a DataText, a DataMemo stb.), vagyis - a Delphi szóhasználata szerint - adatfüggők.

Alapkomponenesek

A Standard eszköztár hét komponensből áll: Text (Szöveg), Memo (Jegyzet), Section (Szakasz.), Bitmap (Bitkép), Metafile (MetaFájl), FontMaster (Betűkezelő) és PageNumlnit (Lapsorszám). A jelentések tervezése során ezeket a komponenseket általában fel is használjuk.

A *Text komponens* segítségével egysornyi szöveget jeleníthetünk meg a jelentésben. Leginkább egy címkéhez hasonló, amely egyszerű szöveget tartalmaz. Amikor elhelyezzük a jelentésben, a mező körül egy keret jelenik meg, amely a komponens határait jelzi.

A *Memo komponens* hasonló a Text komponenshez, az utóbbival ellentétben azonban több sornyi szöveget is tartalmazhat.

A *Section komponens* a Delphi Panel komponenséhez hasonlóan a komponensek csoportosítására szolgál. A csoportosítás révén lehetőségünk nyílik arra, hogy egyetlen mozdulattal áthelyezzük a csoport teljes tartalmát, így nem kell a komponensek egyesével történő mozgatásával vesződnünk.

A *Bitmap* és a *Metalile komponensek* segítségével képeket helyezhetünk cl a jelentésben. A Bitmap .bmp kíterjesztésű raszteres képek, a Metafile pedig .wmf és .emf kiterjesztésű vektorképek megjelenítését támogatja.

A *FontMaster komponens* segítségével szabványos betűtípusokat állíthatunk össze a jelentés különböző részei (fejléc, törzs és lábléc) számára.

A *PageNumlnit* egy nem látható komponens, amely lehetővé teszi, hogy újrakezdjük a lapszámozást a jelentésen belül.

Rajzoló komponensek

A szabványos komponensekhez hasonlóan a rajzoló komponensek sem kötődnek az adatokhoz. A Rave jelentések háromféle vonalat tartalmazhatnak: tetszőleges irányú, függőleges és vízszintes vonalakat. A geometriai alakzatok között megtaláljuk a négyzetet, a téglalapot a kört és az ellipszist is.

Vonalkód-komponensek

A vonalkód-komponensek segítségével különféle vonalkódokat helyezhetünk el a jelentésben.

Területek és sávok

A területek (Region) a sáv komponensek (Band) tárolására szolgálnak. A terület legegyszerűbb formájában a teljes lap (Page) lehet.

Kétféle sáv létezik:

- *DataBand* Ismétlődő adatokat jelenít meg a DataWiew komponensből. A DataBand általában több DataText komponenst tartalmaz.

- *Band* Fejléc- és lábléc-sávok létrehozására használható. A támogatott fejlécek és láblécek között megtaláljuk a Body (Törzs), a Group (Csoport) és a Row (Sor) típusokat.

Adatfüggő komponensek

A DataBand belsejében különféle adatfüggő Rave komponenseket helyezhetünk el. Ezek közül a legáltalánosabb a *DataText*, amely az adathalmaz egy szövegmezőjének tartalmát jeleníti meg.

A *DataMemo komponens* a DataWiew-ból származó jegyzetmezők megjelenítésére alkalmas, legyen az akár több sornyi is.

A *CalcText komponens* egy adatfüggő összegszámító komponens, kifejezetten összegek kiszámítására és megjelenítésére tervezték.

A *DataCycle komponens* valójában nem más, mint egy láthatatlan adatsáv, amely adatismétlési lehetőségekkel egészíti ki a lapot.

VI. Súgórendszer készítése Delphi alkalmazásokhoz

A komolyabb alkalmazások sok mindenben különböznek egymástól, de egy biztos: nincs köztük olyan, amelyiknek ne lenne súgója *(Help).* A felhasználót a program kezelésére oktató, illetve sok más hasznos információt kínáló súgórendszer megléte az alkalmazásokban alapkövetelmény. A felkínált információanyag jellegétől, illetve a súgó hívási módjától függően a súgórendszer tervezésénél általában a következő lehetőségek közül választhatunk:

A Súgó menün keresztül aktivizálható legfontosabb funkciók:

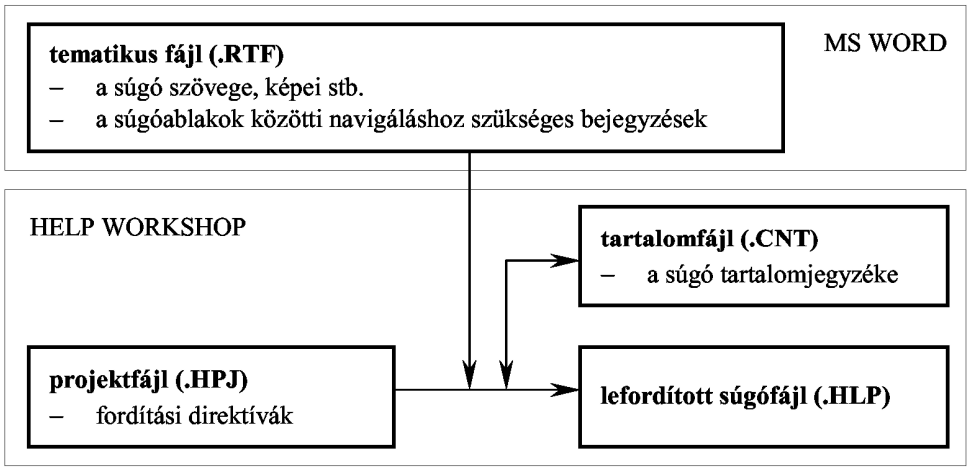
- fő súgóablak megjelenítése (a művelet Windows-ban alapértelmezett gyorsító billentyűje az *<F1>),*
- a kérdéses vezérlőelem mellett kisebb keretekben megjelenő környezet-érzékeny súgó kérése a alakúra változott kurzorral („Mi *ez?"* menüpont megvalósítása,*<Shift+Fl>),*
- az ún. készítői névjegy megjelenítése *(About* párbeszédablak),
- újabban a *World Wide Web-en* elhelyezett, támogató weboldalakra való hivatkozás aktiválása, azaz a rendszerben alapértelmezett webböngésző program elindítása és a kérdéses weboldal megjelenítése.

Egyéb (ritkábban alkalmazott) funkciók:

- környezet-érzékeny súgó megjelenítése az egér jobb oldali gombjának valamelyik vezérlőelemen történő lenyomásakor, illetve – kizárólag párbeszédablakokban – a fejlécsorban látható gomb használatakor,
- különféle multimédiás „oktatótúrákat" tartalmazó állományok lejátszása,
- a program futása során állandóan látható súgóablakok megvalósítása, multimédiás elemekkel tarkítva.

Valamely program súgójának fejlesztésekor két feladatot kell megoldanunk: az ún. súgófájlok elkészítését, és a súgókban tárolt információk programból való megjelenítését.

Az alkalmazásokban a Delphi rendszer a hagyományos formátumú súgófájlok használatát támogatja.



Súgófájl (.hlp) írása

A súgófájlok készítése két lépésben történik. Először egy szövegszerkesztőben (pl. *Microsoft Word)* létre kell hozni egy *Rich Text* formátumban *(.rtf)* elmentett ún. tematikus fájlt, amely az alkalmazásunk súgójának szövegét, képeit stb. tartalmazza. Ezután ezt az állományt le kell fordítani kész súgófájllá *(.hlp),* például a telepített Borland Delphi rendszer *Help\Tools* alkönyvtárában található ***Microsoft Help Workshop*** alkalmazás *(hcw.exe)* segítségével. Ehhez azonban először meg kell szerkeszteni a súgónk ún. projektfájlját *(.hpj),* amelyben a fordításhoz szükséges bejegyzéseket és direktívákat adjuk meg. A *Help Workshop* segítségével ez a szöveges állomány pillanatok alatt elkészíthető.

Ha azt szeretnénk, hogy a súgónk a Windows-ban megszokott módon, háromfüles párbeszédablakban kínálja fel tartalmát, akkor szükségünk van egy un. tartalomfájlra *(.cnt)* is, amelyet szintén a *Help Workshop* alkalmazás segítségével állíthatunk elő. A tartalomfájl leírja a súgó párbeszédablak első, *Tartalom* lapjának struktúráját. Ha nem hozunk létre *.cnt* állományt, vagy enélkül fordítjuk a súgófájlt, a súgó párbeszédablaka csak két lapot fog tartalmazni *(Tárgymutató* és *Keresés* lap).

Tematikus állomány létrehozása

A munkát azzal kezdjük, hogy begépeljük a súgó fejezeteinek szövegét valamilyen szövegszerkesztő segítségével. Gépelés közben a szöveget úgy tagoljuk összefüggő részekre (ún. súgófejezetekre, illetve lapokra), hogy a fejezetek közé oldaltörési jelet illesz-

tünk (a Word szerkesztőben például a *Beszúrás* menün keresztül). A súgó hívásakor egyszerre mindig csak egy súgófejezetet (egy lap tartalmát) jelenít meg a Windows.

A súgófejezetek azonosítására az ún. fejezetazonosító szolgál (szünetjelek nélküli karakterlánc), amely a súgófejezet saját egyedi hivatkozási neve. A fejezetazonosítók megadása lábjegyzetben történik, a # jelet követően. (A lábjegyzetben - bonyolultabb súgó esetében - más információkat is hozzárendelhetünk a súgófejezetekhez, a *K,* +, / stb. jelek után.)

A fejezetazonosító beillesztéséhez a szövegkurzort a fejezet elejére kell állítanunk, majd a Word szövegszerkesztő *Beszúrás* menüjéből ki kell választanunk a *Hivatkozás* | *Lábjegyzet* menüpontot, és a megjelenő párbeszédablakban meg kell adnunk a # jelet mint egyedi jelölést.

A kész tematikus fájlt RTF-állományként kell elmentenünk *(Fájl* | *Mentés másként),* fájltípusként kiválasztva a *Rich Text* formátumot.

A projektállomány létrehozása, és a súgófájl fordítása

A tematikus állomány *(.rtf)* létrehozása után következő lépés a súgó projektfájljának elkészítése és a súgófájl fordítása. Ehhez nagy segítséget nyújt a *Help Workshop* alkalmazás. A súgó projektállománya a fordítóprogram számára szükséges információkat tartalmazza: milyen tematikus állományokból, milyen jellemzők hozzáadásával, hogyan tömörítve stb. épüljön fel a végleges súgófájl.

Környezet-érzékeny súgó készítésekor nagyon fontos lépés az un. kontextszámok megadása. Ezeket a számokat a programban arra használjuk, hogy a különböző vezérlőelemekhez megfelelő súgófejezeteket rendeljünk hozzá. A kontextszámokat a vezérlők ***HelpContext*** tulajdonságában kell beállítani.

Környezet-érzékeny súgó készítésekor a kontextszámok megadása után az állományok végleges súgófájllá *(.hlp)* való fordítása következik, amit a *„Save and Compile"* gombon kattintva indíthatunk el.

A tartalomfájl létrehozása

Alkalmazásunk - a környezet-érzékeny súgón kívül - rendelkezhet bonyolultabb, kézikönyv jellegű súgóval is, melyet az alkalmazások többsége külön *Súgó* menün, illetve azon belül a *"Tartalom és Index"* menüponton keresztül aktivizál. A súgó „belépési pontja" ebben az esetben általában egy háromoldalas párbeszédablak, amelyből témaköröket vá-

laszthatunk ki. A párbeszédablak első, *Tartalom* oldalának szerkesztését a *Help Workshop* alkalmazás segítségével végezhetjük.

A tartalomfájl *(.cnt)* írását azzal kezdjük, hogy a *Help Workshop* ablakában látható legfelső *Edit* gombon kattintva, megadjuk a súgóablak címsorában megjelenő szöveget *(Default title),* valamint az alapértelmezés szerinti súgófájl nevét *(Default filename).*

A tartalomfájl bejegyzéseit (a súgóablak első lapján megjelenő témacímek) az *„Add Above"* és az *„Add Below"* nyomógombok segítségével adhatjuk meg. Az ekkor megjelenő *„Edit Contents Tab Entry"* párbeszédablakban először is jeleznünk kell a bejegyzés típusát *(Heading* - egy bejegyzéscsoport címe, *Topic* - egy súgófejezetre hivatkozó bejegyzés). Egy csoportnév létrehozásához, a bejegyzések megadása során, be kell jelölnünk a *Heading* választógombot (a csoport a *Topic* választógomb bejelölésével megadott bejegyzéseket, vagyis a témaköröket fogja össze).

Súgóablakban mindegyik csoportnév mellett egy könyv alakú ikon jelenik meg, amelyre rákattintva „kinyitjuk" a könyvet, azaz megjelenítjük a csoportba gyűjtött bejegyzéseket.

A kész tartalomfájlt *(.cnt)* ugyanabba a mappába kell elmentenünk *(File | Save as),* ahol a projektfájl is található. Ha a *.hlp* állomány nevével megegyező néven tároljuk a *.cnt* állományt is, akkor a súgófájl megnyitásakor a Windows Súgórendszere megkeresi az azonos nevű tartalomfájlt, és megjeleníti annak tartalmát a súgóablak *Tartalom* lapján.

Tematikus állományok készítése bonyolultabb súgókhoz

Kézikönyvszerű súgók esetén a tematikus fájlok készítését a környezet-érzékeny súgóhoz hasonló módon kezdjük (egymástól oldaltörési jellel elválasztott súgófejezetek, mindegyik saját, a lábjegyzetben a # jellel megadott fejezetazonosítóval). Ebből kiindulva a súgófájlunkba még sok egyéb lehetőséget is beépíthetünk.

A fejezetek közötti navigáláshoz, az „ugrások" végrehajtásához un. hivatkozásokat kell készítenünk a súgó szövegének írása során. Ehhez a hivatkozásként használni kívánt szövegrészt dupla aláhúzással vagy pontozott vonallal (Word | *Formátum* | *Betűtípus* | *Aláhúzás típusa:* dupla, illetve pontozott) kell kiemelnünk, majd ezután rejtett *(Különlegességek | Rejtett)* betűkkel, szóközök nélkül meg kell adnunk a hivatkozott súgófejezet fejezetazonosítóját.

A dupla aláhúzást akkor alkalmazzuk, ha a hivatkozásra való kattintás után a súgóablakban a jelzett fejezetazonosítójú súgófejezetnek kell megjelennie. Pontozott vonallal

való jelölés olyan hivatkozást eredményez, amelyre kattintva a megadott fejezet szövege egy kisebb keretben jelenik meg a hivatkozás mellett. Ezt a hivatkozástípust általában különféle fogalmak magyarázatához használjuk.

Ha el kívánjuk kerülni, hogy a súgó megjelenítésekor más színű legyen a hivatkozások szövege, mint a többi szöveg, akkor a fejezetazonosító előtt - ugyancsak rejtett betűkkel - meg kell adnunk a * jelet is.

Súgófájlok hívása alkalmazásokból

A kész súgófájlok által tárolt információkat többféleképpen jeleníthetjük meg a programunkban.

A súgófájl nevének megadása

Az első lépésben meg kell adnunk az alkalmazás által használt súgófájl nevét. Ezt megtehetjük az alkalmazásobjektum ***HelpFile*** tulajdonságában:

```
Application.HelpFile := 'ÁLTALÁNOS.HLP';
```

illetve - a környezet-érzékeny súgó esetén - a form ***HelpFile*** tulajdonságában:

```
Forml.HelpFile := ExtractFilePath(Application.ExeName) + 'KONTEXT.HLP' ;
```

A tulajdonságokat fejlesztési időben is beállíthatjuk: ha űrlapra vonatkoznak, akkor az objektum-felügyelőben, ha pedig az alkalmazásra - akkor a projekt tulajdonságlapjain *(Project | Options | Application | Helpfile).*

Környezet-érzékeny súgó esetén a súgófájl nevének beállítása után meg kell adnunk az űrlapon elhelyezett vezérlőelemek, illetve menük ***HelpContext*** tulajdonságában azt a kontextszámot, amellyel a megfelelő súgófejezetre hivatkozhatunk.

Ezzel az egyszerű megoldással az alkalmazásunk máris rendelkezik saját környezetérzékeny súgóval, melynek a futó alkalmazásból való hívásához az egérmutatót valamelyik vezérlőelemre kell mozgatnunk, majd meg kell nyomnunk az <*F1*> billentyűt.

A súgó párbeszédablakának megjelenítése

A kézikönyvszerű súgó hívását legtöbbször a programunk *Súgó* menüjében létrehozott *"Témakörök és index"* menüpont működtetésével, illetve egy külön nyomógomb lenyomásával érjük el.

VII. A program ismertetése

Feladatspecifikáció

A program célkitűzései:

> Törzsadatok nyilvántartása, karbantartása
> Vételezések kezelése
> Bizonylatok nyilvántartása, nyomtatása
> Statisztikák készítése

Az alkalmazással szembeni elvárás, hogy a program alkalmas legyen a anyagvételezés nyilvántartására. Egy cikk kivételezésekor rögzíteni kell a cikkszámot, a cikk megnevezését, a mennyiséget, és mennyiségi egységet, felhasználás okát, dátumát, vételező nevét. A kiadási eseményeket nyilvántartásba kell venni. Lehetővé kell tenni az adatok módosítását, törlését, új adatok felvételét, az adatbázisban való keresést, valamely szempont szerinti lekérdezést, bizonylatok ill. lekérdezések nyomtatását.
A programok készítésének első és legfontosabb fázisa a tervezés. Fontos szempont a programtervezésnél, hogy a rendszer átlátható és könnyen bővíthető legyen.

Az adatbázis

A program egy BDE-re épülő lokális adatbázis-kezelő alkalmazás, melyben a vételezés adatait és eseményeit DBaseIV típusú táblákban vannak tárolva. A BDE a táblákat az adatbázishoz rendelt alias-on keresztül éri el. Az alias paraméterei a következők:

> Type: Standard
> Default driver: DBase
> Path: C: \ Munka \ Data (ami függhet a telepítéstől)

A programban használt táblák szerkezete:

FH.dbf	Mezőnév	Típus
	▪ NEV	Character(50)
	▪ TORZSSZAM	Character(7)
	▪ LOGIN	Character(10)

- SZOLGH Character(50)
- JOG Number(1,0)
- SZH_KOD Number(5,0)

KOD.dbf

- KOD Character(8)
- T_TIP_LEIR Character(80)
- T_OK_LEIR Character(60)

JAV_KOD.dbf

- JAV_NEM Character(3)
- KOD Character(2)

AKJEGYES.dbf

- SORSZAM Number(5,0)
- BIZ_SZAM Character(7)
- BIZ_NEME Number(2,0)
- DATUM Date
- TORZSSZAM Character(7)
- KOD Character(8)
- CIKKSZAM Number(10,0)
- MEGNEVEZES Character(40)
- MENNYISEG Number(5,0)
- ME_EGYSEG Character(20)
- JAV_NEM Character(3)
- GPR_ALFEL Character(25)
- GPR_SZAM Character(25)
- MUNKASZAM Number(10,0)
- RAKTAR_KOD Character(5)
- K_HELY Character(4)
- ELLENORZES Number(1,0)
- NYOMTATOTT Number(1,0)

VETELEZETT.dbf

- SORSZAM Number(5,0)
- V_MENNYISEG Number(5,0)
- V_CIKKSZAM Character(10)
- V_BIZ_SZAM Character(7)

CIKKTORZS.dbf

- CIKKSZAM Character(10)
- LEIRAS Character()150
- ME Character(5)
- AKT_AR Number(10,2)

ME.dbf

- ME Character(5)
- M_EGYSEG Character(20)

GIRPR.dbf

- GPR_SZAM Character(25)
- GPR_ALFEL Character(25)
- UTK Character(4)
- SZHKOD Character(5)

A táblák funkciói:

FH.dbf → A felhasználók adatait tartalmazza

KOD.dbf → A vételezés okának szöveges tárolása

JAV_KOD.dbf → A mozdony/személykocsi javítási nemeket és kódokat tárolja

AKJEGYES.dbf → A kiírt anyagok nyilvántartása

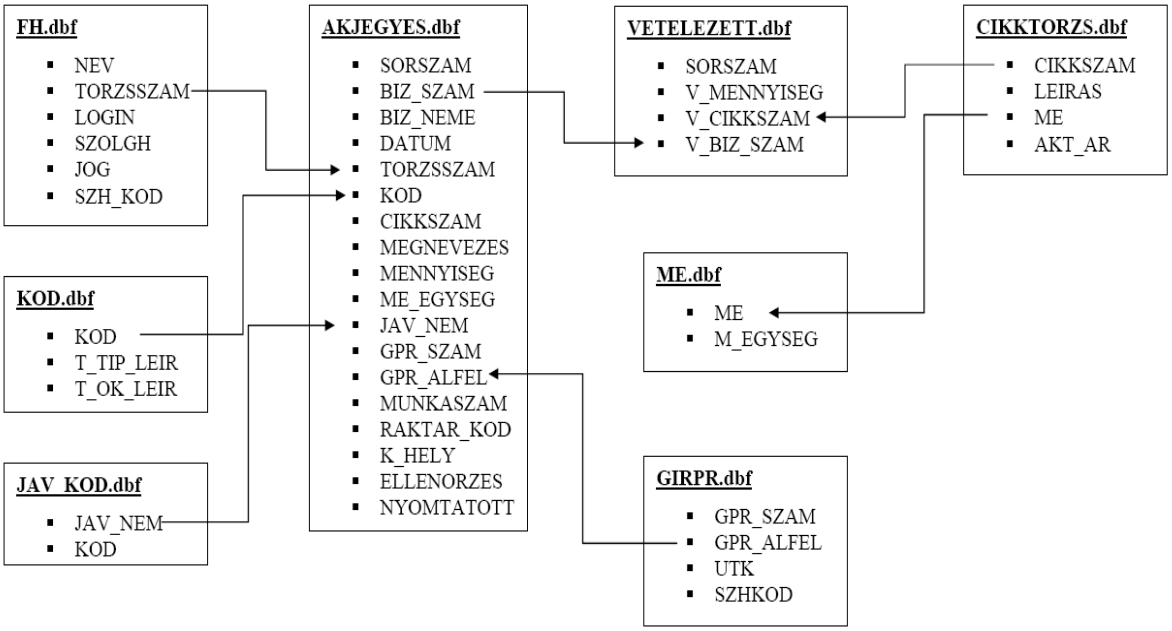
VETELEZETT.dbf → A ténylegesen kivételezett anyagok nyilvántartása

CIKKTORZS.dbf → A cikk adatainak tárolása

ME.dbf → A mennyiség egységek nyilvántartása

GIRPR.dbf → Mozdony/személykocsi javítási projektek és feladatok

Táblák közötti kapcsolatok



A program telepítése

A program az InstallShield Express nevű segédprogrammal létrehozott telepítő lemez segítségével installálható.

A Delphi programcsomag tartalmaz egy InstallShield Express nevű programot, mellyel egyszerűen és gyorsan készíthetünk komplett telepítőkészletet. Kifejezetten Delphi-hez készült, így támogatja a BDE-t, mely az adatbázist kezelő programokhoz elengedhetetlen. Az InstallShield Express nem a Delphi része, de a Delphi telepítő CD-jén önálló programként megtalálható és installálható.

A program telepítése során a felhasználó beállíthatja a kívánt telepítési útvonalat, majd a Start menüben létrejön egy mappa, mely a programindító parancsikont tartalmazza.

A telepítés után a program azonnal indítható.

A program kezelése

Bejelentkezés

Bejelentkező képernyő, a dolgozó felhasználó nevével és vasúti törzsszámával. Az adatok beírása után a Belépés gombra kattintva indul a program. Helytelen adat megadása esetén figyelmeztetés.

A program indításához szükséges felhasználó név: karesz; törzsszám: 4013769.

Névjegy

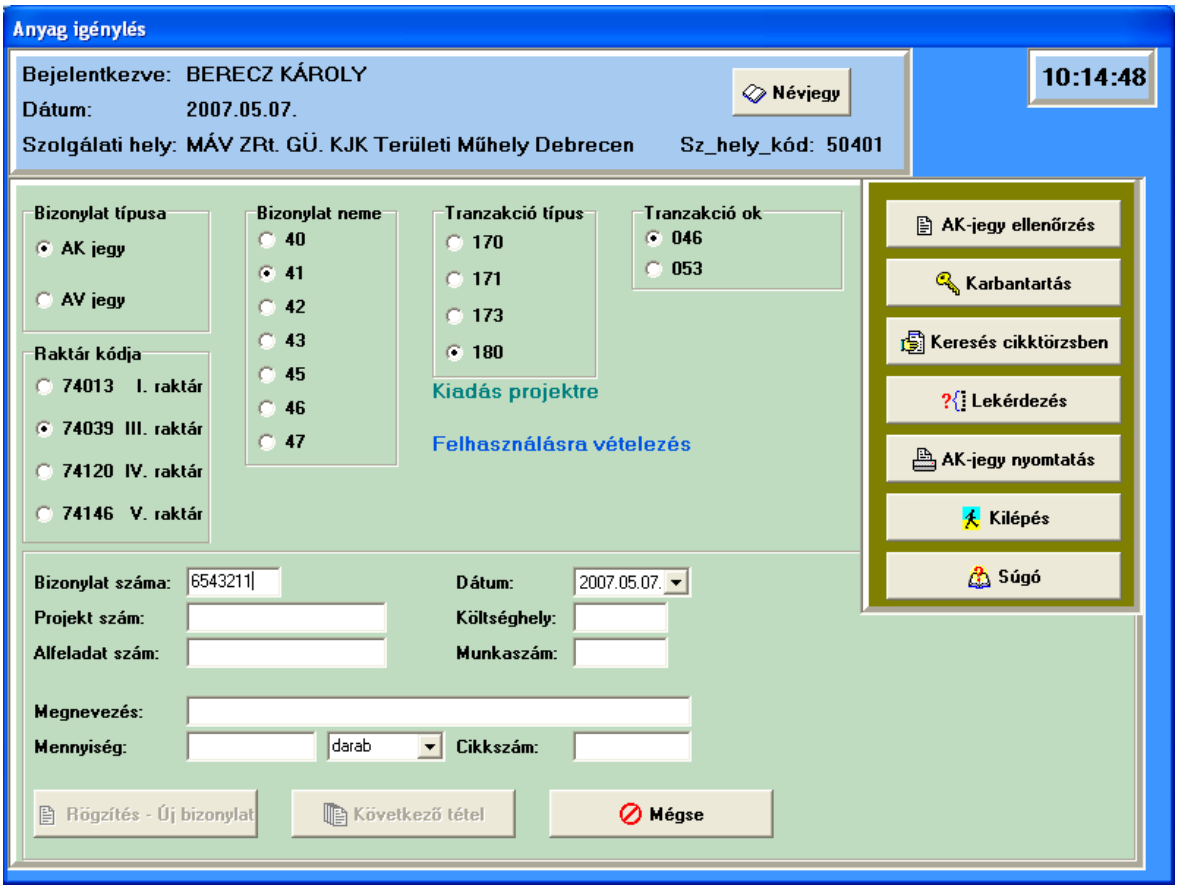
A program verziószámát, készítőjét valamint elérhetőségét tartalmazza.

Az e-mail címre klikkelve indul az alapértelmezett levelező program.

Anyagigénylés

A program elsődleges munkafelülete. Tartalmazza a bejelentkezett felhasználó nevét, szolgálati helyét, az aktuális dátumot, valamint a pontos időt.

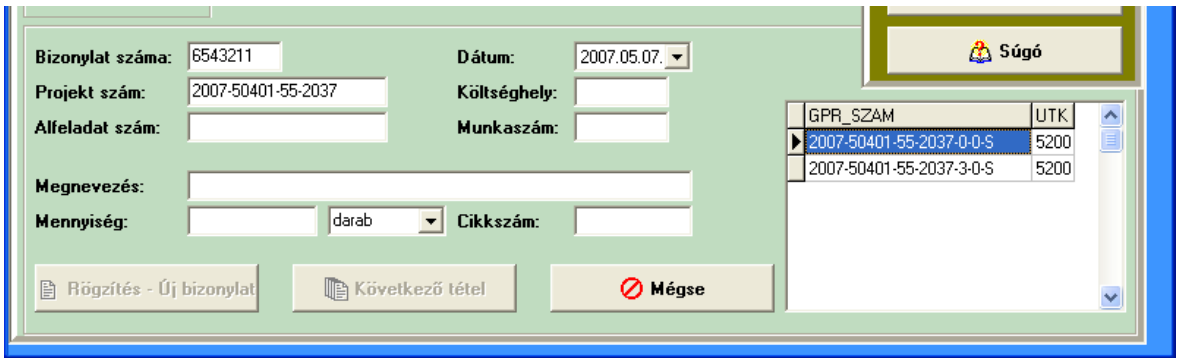
Az AK-jegy rögzítés folyamata: A bizonylat típusának kiválasztása. Ennek megfelelően jelenik meg a bizonylat neme. Választást követően megjelenik a tranzakció típus, majd a tranzakció ok. Következő választási lehetőség az adott szolgálati helyhez kapcsolódó raktárak és kódjaik. Eközben szövegesen is megjelenik a típusának és okának szöveges leírása. A raktár megjelölése után megjelenek az AK-jegy kitöltendő részei.



A beviteli mezők között továbblépést az Enter és a Tab billentyű, valamint az egérrel való kattintás biztosít.

Beviteli mezők:

- *Bizonylat száma*: 7 jegyű, numerikus kód, az AK-jegy száma. Már rögzített bizonylat esetén figyelmeztetés.
- *Projektszám*: A javítási projekt rögzítésére szolgáló, karakteres mező. Gépelés közben megjelenő panelen felkínálja a beírt projekttel megegyező kezdetűt. Az aktuális rekordot a ►jelzi. Kiválasztani az Enter billentyűvel, vagy az egérrel lehet. Panelen belüli mozgás megvalósítható a ↓ és a ↑ mutató kurzorbillentyűkkel.



Nem létező projekt esetén felajánlja a felvitel lehetőségét.

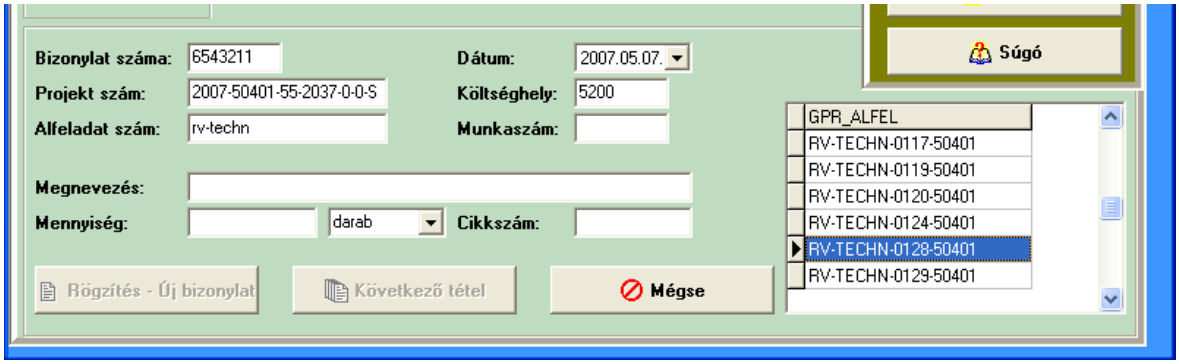
Ha az adatbázisban szerepel az adott projekt kitöltésre kerül a költséghely mező, ellenkező esetben kézi megadás szükséges.

Projektszám felépítése:

2007-50401-55-2037-0-0-S

- S: Jármű fajtát jelölő kód (S: vontatott, M: vontatójármű)
- 0: Segédkód
- 0: Felújított jármű kódja
- 2037: Jármű sorozatszáma
- 55: Honos vasút kódja (55: MÁV)
- 50401: Szogálatihely kód
- 2007: Évszám

- *Feladatszám*: Az elvégzendő feladat pontosabb leírására szolgáló, karakteres mező. Beíráskor megjelenő panelen felkínálja a beírt feladattal megegyező kezdetűt. A megjelenő feladatok a projektszám mezőben megjelenő érték alapján kerülnek



szűrésre.

Az aktuális rekordot a ►jelzi. Kiválasztani az Enter billentyűvel, vagy az egérrel lehet. Panelen belüli mozgás megvalósítható a ↓ és a ↑ mutató kurzorbillentyűkkel. Nem létező feladat esetén felajánlja a felvitel lehetőségét.
Ha az adatbázisban szerepel az adott feladat kitöltésre kerül a munkaszám mező, ellenkező esetben kézi megadás szükséges.

A feladatszám felépítése:

RV-TECHN-0128-50401

- 50401 — Javítást végző szolgálati hely kódja
- 0128 — A jármű sorszáma
- TECHN — Javítási mód
- RV — Javítási nem

Az Enter billentyű lenyomásakor, ha a vágólapra adat kimásolása történt a Keresés MÁV cikktörzsben panelen, felajánlja a beillesztés lehetőségét.

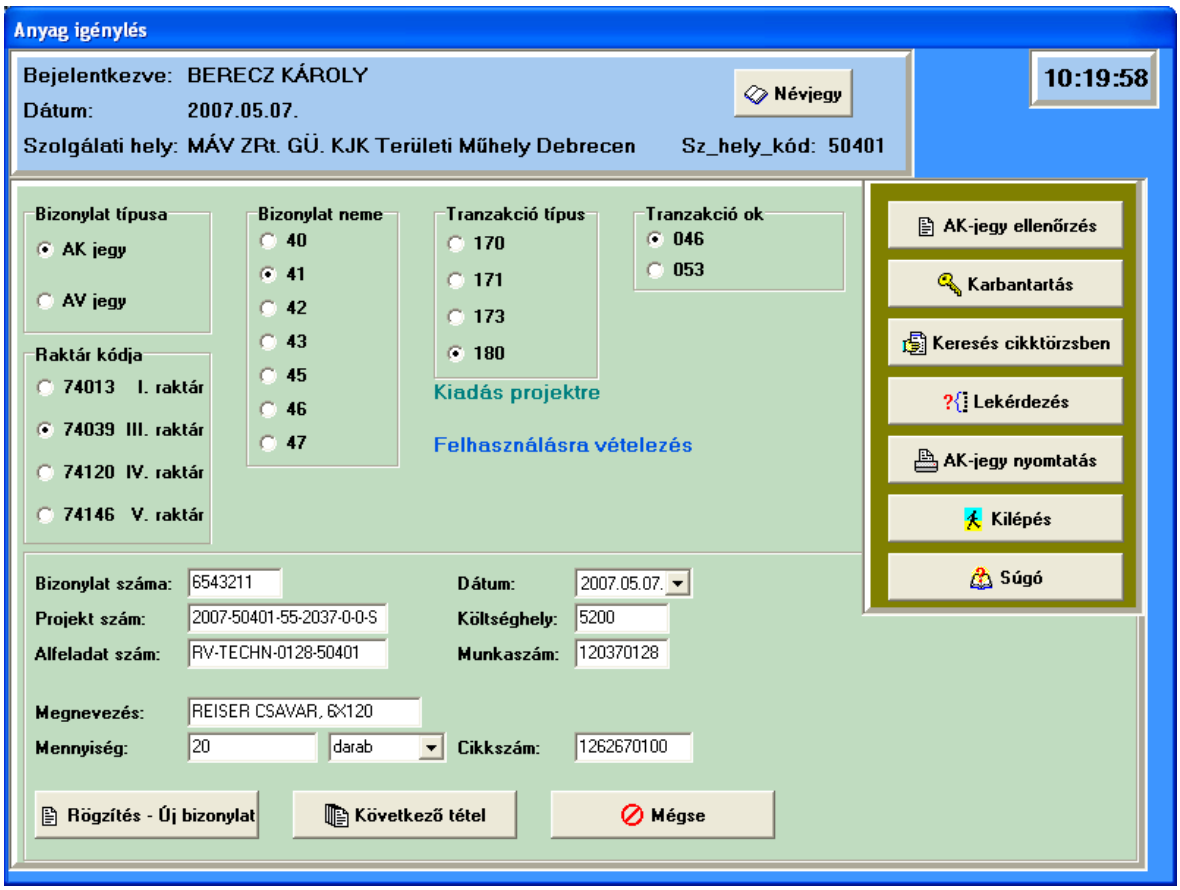
- *Dátum*: A vételezés dátumát jelöli, alapértelmezett az aktuális dátum.
- *Költséghely*: 4 jegyű, numerikus kód. A projekthez tartozó költséghelyet jelöli. Létező projekt esetén automatikus kerül kitöltésre, szükség esetén módosítható.
- *Munkaszám*: 9 jegyű, numerikus kód. A feladathoz tartozó munkaszámot jelöli. Létező feladat esetén automatikus kerül kitöltésre, szükség esetén módosítható.

- *Megnevezés*: A vételezett anyag megnevezése. Karakteres mező. Kitöltése lehet kézi, illetve kikereshető a MÁV cikktörzsből.
- *Cikkszám*: Az anyag beazonosítására szolgál. 10 jegyű numerikus kód. Kitöltése nem kötelező.
- *Mennyiség*: A kivételezendő mennyiség. Nem féltetlenül egyezik meg a ténylegesen kivételezett mennyiséggel.
- *Mennyiségegység*: A kiírásra kerülő anyagféle mennyiségegysége. Legördülő listából történik a kiválasztása.

A beviteli mezők kitöltése után aktívvá válnak a rögzítésre szolgáló nyomógombok.

- *Rögzítés - Új bizonylat*: A bevitt adatok rögzítésre kerülnek, új bizonylat rögzítése kezdhető meg.
- *Következő tétel*: Ugyanazon bizonylatra rögzíthető a következő kivételezendő anyag. Egy bizonylatra maximum 7 tétel rögzíthető.

Rögzítés nélküli visszalépésre szolgál a Mégse gomb.

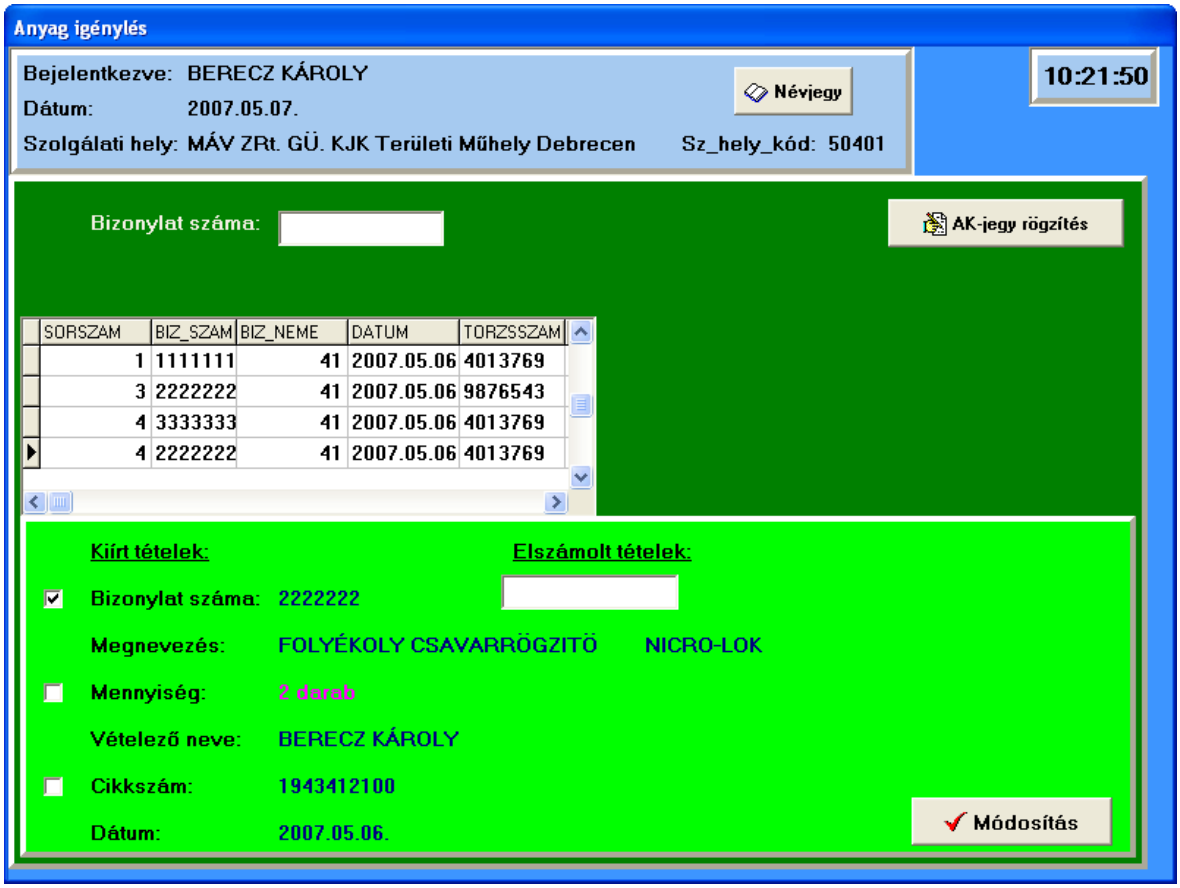


AK-jegy ellenőrzés

A bizonylat számának megadásakor megjelenek az addig rögzített és még nem ellenőrzött bizonylatok. Az aktuális rekordot a ►jelzi. Kiválasztani az Enter billentyűvel, vagy az egérrel lehet. Panelen belüli mozgás megvalósítható a ↓ és a ↑ mutató kurzorbillentyűkkel.

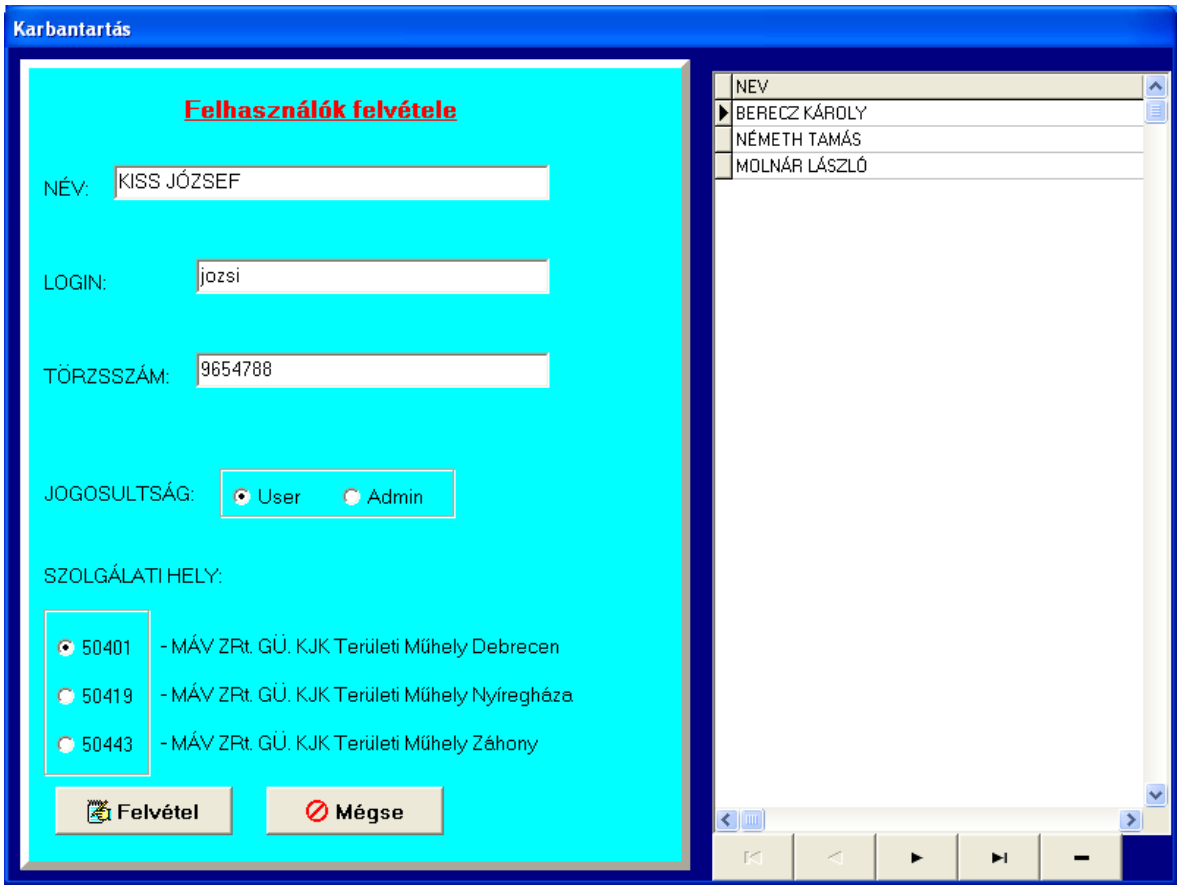
A megfelelő számú bizonylat kiválasztásakor a megjelenő panelen leolvasható a kiírt tétel. A tételek előtti jelölőnégyzet segítségével lehet az adatokat módosítani. Módosítható a bizonylat száma, megadható a ténylegesen kiadott mennyiség, és a tényleges cikkszám is. A módosított adat más színnel kerül megjelölésre. Az adatbázisban tényleges rögzítés a Módosítás nyomógomb segítségével történik, melynek tényét a megjelenő táblázatban lehet megtekinteni.

A felvitelhez való visszatérésre az AK-jegy rögzítés címkéjű nyomódomb szolgál.



Karbantartás

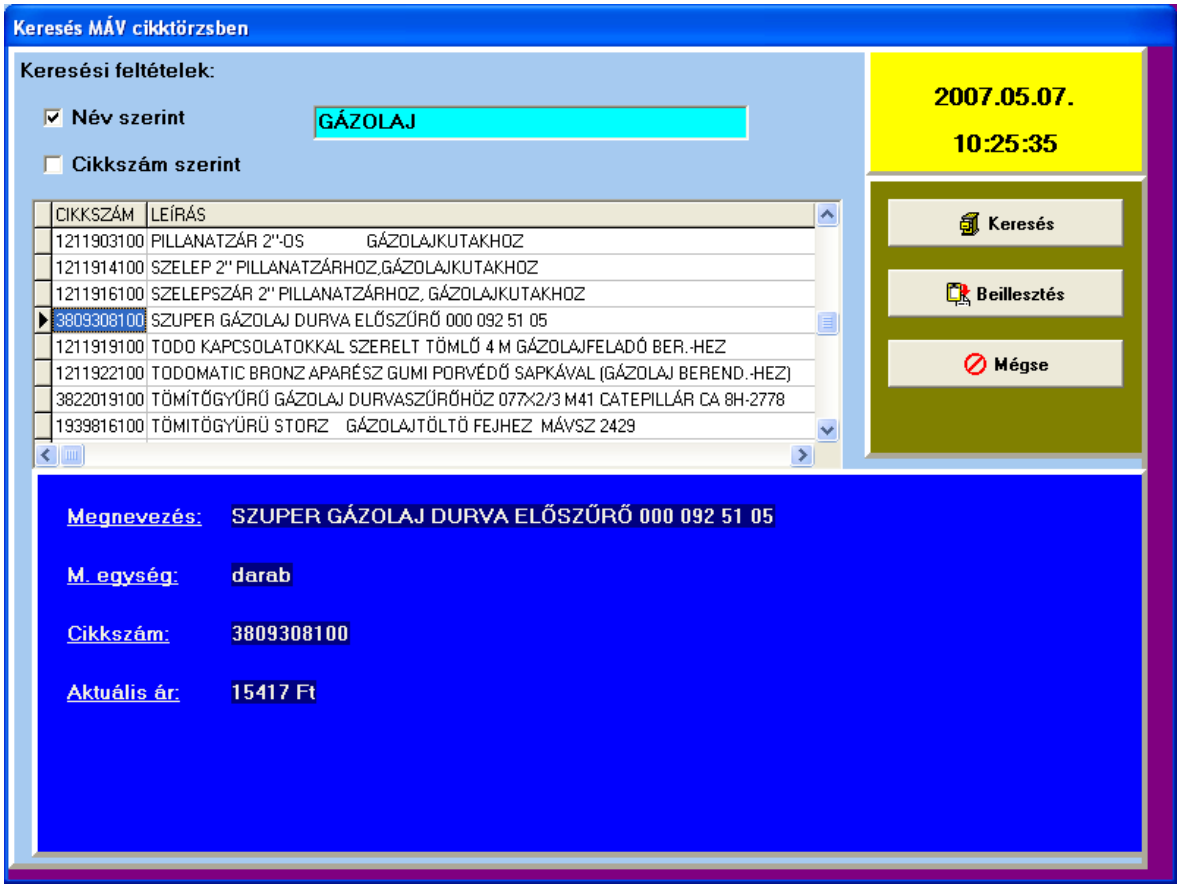
Felhasználók felvételére szolgáló panel. Rögzítésre kerül a dolgozó neve, felhasználóneve, vasúti törzsszáma. Jogosultság alapján van lehetősége lekérdezések futtatására, valamint a karbantartás elvégzésére. A szolgálati hely kódjának megadása után válik aktívvá a Felvétel nyomógomb. A rögzítés megszakítása a Kilépés gombbal történik. Azonos törzsszámmal rendelkező dolgozó rögzítése nem megoldható. Az eddig felvitt dolgozók névsorát a mellékelt táblázat tartalmazza.



Keresés MÁV cikktörzsben

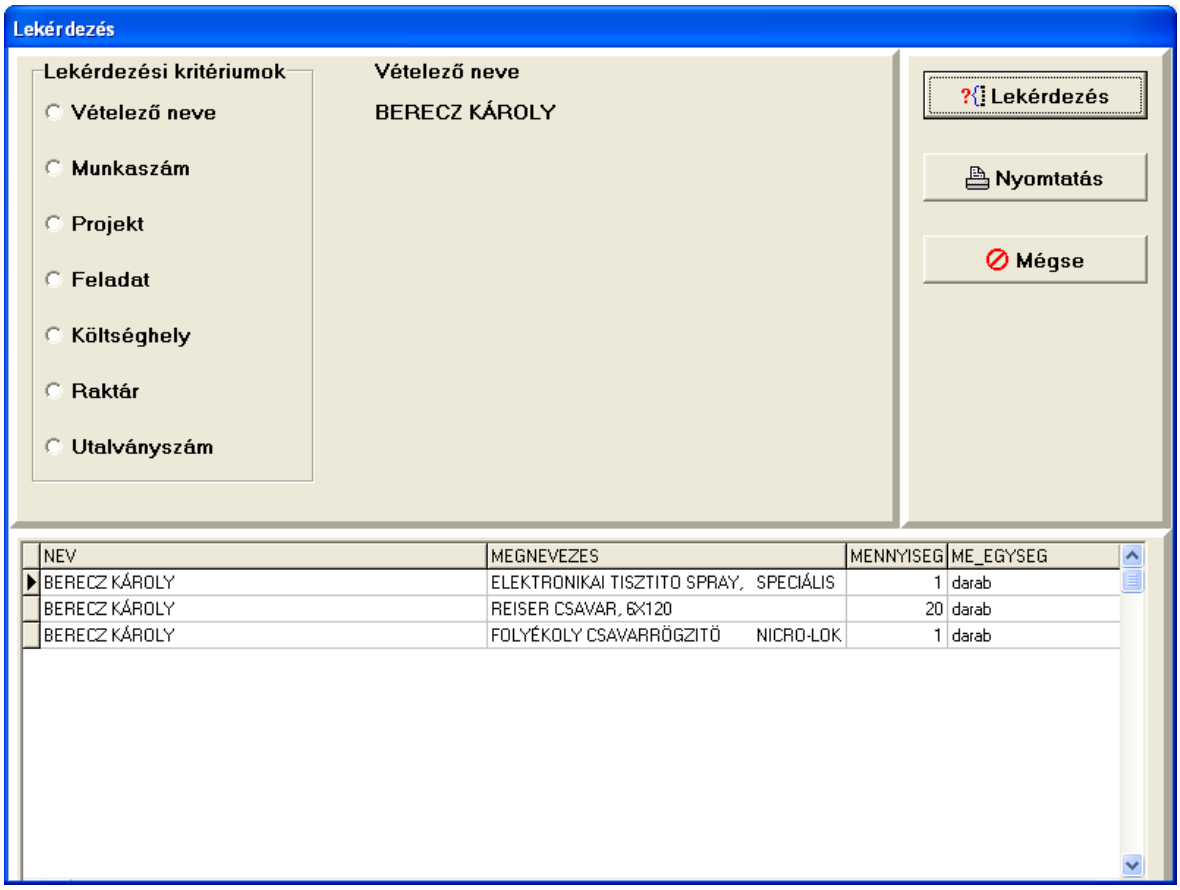
A MÁV cikktörzsben való keresésre szolgáló panel. A keresési feltétel kiválasztásra szolgál a két jelölőnégyzet. Név szerinti keresés esetén a cikktörzs leírás, míg cikkszám szerintinél a cikktörzs cikkszám mezőjében történik rész sztring keresés. A keresés indítása a Keresés gombbal történik. A megjelenő táblázatban lehet kiválasztani a nekünk megfelelő tételt, amikor is egy panelen megjelennek a jellemző paraméterek, illetve aktívvá válik a Beillesztés/Másolás vágólapra nyomógomb. A nyomógomb kettős funkciót lát el attól függően, hogy az AK-jegy rögzítés panelen a vételezett anyag megnevezése aktív-e. Amennyiben aktív beillesztés történik a megfelelő helyre, ellenkező esetben csak a vágólapra történik másolás.

Az ablak bezárására a Vissza gomb szolgál.



Lekérdezés

Különböző összegző lekérdezésre szolgáló panel. A lekérdezési kritérium kiválasztása, majd megadása után a Lekérdezés nyomógomb megnyomásával végezhető el az összegzés.



A megjelenő táblában látható az eredmény. Sikeres lekérdezés esetén lehetőség van az eredmény kinyomtatására. A Nyomtatás nyomógomb megnyomásával aktivizálódik a Nyomtatás panel, ahol kiválasztható, hogy nyomtatni, megtekinteni illetve fájlba menteni szeretnénk a lekérdezés eredményét.

Visszalépésre a Vissza nyomógomb szolgál.

A program által generált jelentés

Jelentés

Ak szám	Név	Megnevezés	Mennyiség	Raktár	Dátum
1111111	BERECZ KÁROLY	ELEKTRONIKAI TISZTITO SPRAY, SPECIÁLIS	1 darab	74039	2007.05.06

Projektszám	Feladatszám	Munkaszám	Költséghely	
2007-50401-55-2037-0-0-S	K2-EGYEB-0012-50401	220370012	5200	

Ak szám	Név	Megnevezés	Mennyiség	Raktár	Dátum
3333333	BERECZ KÁROLY	REISER CSAVAR, 6X120	20 darab	74039	2007.05.06

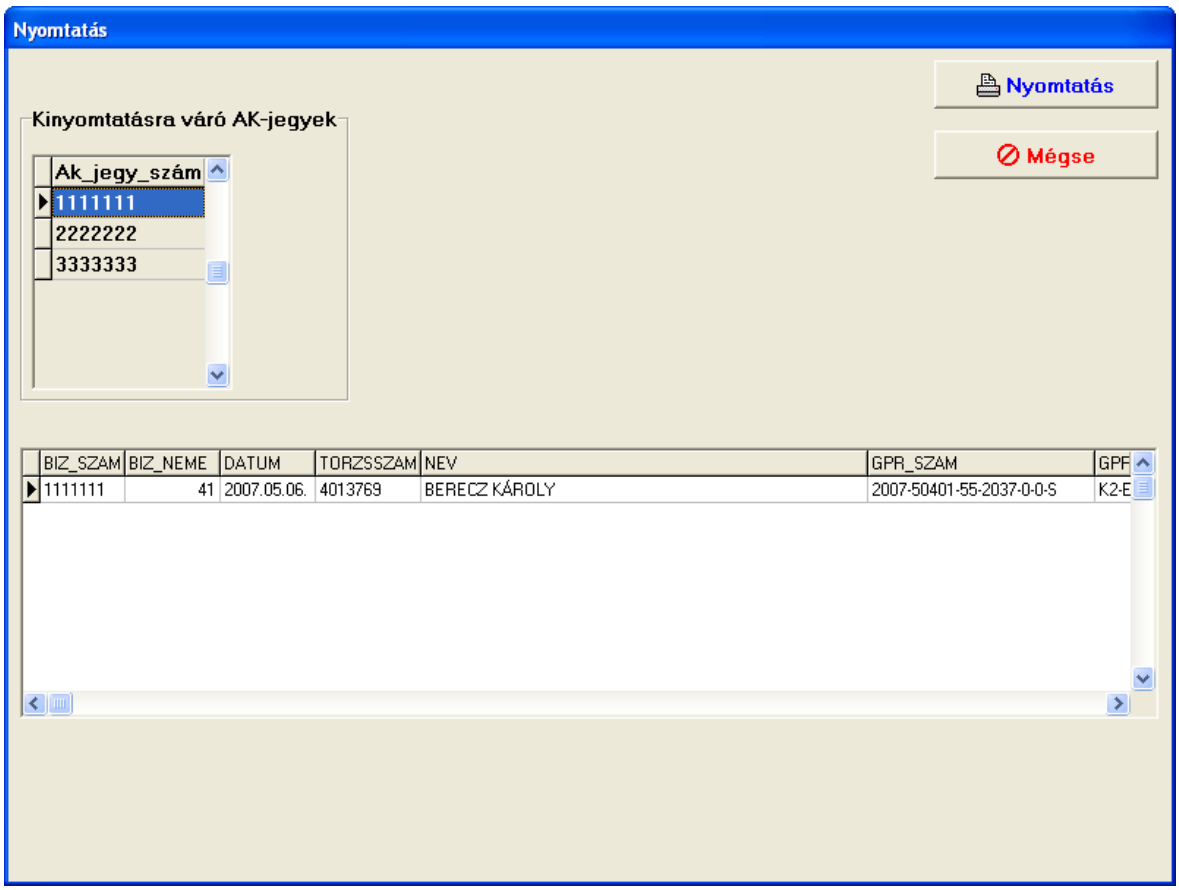
Projektszám	Feladatszám	Munkaszám	Költséghely	
2007-50401-55-2005-0-0-S	FF-EGYEB-0037-50401	720050037	5200	

Ak szám	Név	Megnevezés	Mennyiség	Raktár	Dátum
2222222	BERECZ KÁROLY	FOLYÉKOLY CSAVARRÖGZITŐ NICRO-LOK	1 darab	74039	2007.05.06

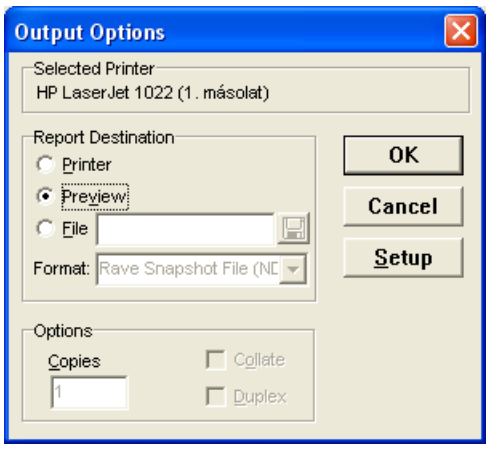
Projektszám	Feladatszám	Munkaszám	Költséghely	
2007-50401-55-2037-0-0-S	FF-EGYEB-0036-50401	720370036	5200	

Nyomtatás

A kitöltött AK-jegy kinyomtatására szolgáló panel. A táblázatból kiválasztható a még ki nem nyomtatott bizonylat száma. Az egérrel történő dupla klikk után megjelenő táblában ellenőrizhetők az adatok.



A Nyomtatás gombra klikkelve kiválaszthatjuk, hogy nyomtatni, megtekinteni illetve fájlba menteni szeretnénk a bizonylatot.



Visszalépésre a Mégse/Cancel gomb szolgál.

A program által kinyomtatott anyagvételezési bizonylat

Magyar Államvasutak ZRt. **Selejtezhető: 2012 év**

Cikk kiadás, visszavételezés, értékesítés

1. példány

Bizonylat száma: 1234568	**Név:** BERECZ KÁROLY **Törzsszám:** 4013769
Bizonylat neme: 40	**Szolg. hely:** - MÁV ZRt. GÜ. KJK Területi Műhely Debrecen 50401
Projekt: 2007-50401-55-2037-0-0-S	**Tranzakció tipus:** Szakanyag kivételezése gyártáshoz és javításhoz
Feladat: RV-TECHN-0128-50401	**Tranzakció ok:** Vételezés javításra
Költséghely: 5200	**Munkaszám:** 120370128 **Raktár:** 74146

Cikkszám	Mennyiség	Menny. egység	Kiadott mennyiség	Me egys.	Cikkszám
1262670100	50	darab			

Megnevezés: REISER CSAVAR, 6X120

2007.05.01.	2007..... hó .....-n.	2007..... hó .....-n.	2007..... hó .....-n.
Utalványozta	**Kiadta**	**Átvette**	**Ellenőrizte**
PH.	PH.	PH.	PH.

0 000012 345687

Anyagvételezés nyilvántartó rendszer Készült: 2007. Verzió: 2.0.0

A programban használt kódok leírása

Vételezés típusának leírása

161	Túlvételezés visszaadása
162	Üzem- és fűtőanyag visszavételezése
163	Ruházat visszavételezése
164	Bontás visszнyereménye
166	Selejtezés visszнyereménye
168	Visszavételezés projektről
170	Anyag felhasználás
171	Üzem- és fűtőanyag felhasználás
172	Ruházat felhasználás
173	Saját termelésű készletek felhasználása
176	Kiadás bérmunkára
180	Kiadás projektre
196	Első kiszerelésre vételezett anyag átadása, első kiszerelésű készletcsoportba
197	Első kiszerelésre vételezett anyag visszamozgatása készletcsoportba
280	Szakanyag kivételezése gyártáshoz és javításhoz

Vételezés okának leírása

033	Anyagselejtezés visszнyereménye
036	Bontás visszнyereménye
038	Túlvételezés visszaadása
039	Egyen- és formaruha visszнyereménye
040	Munka- és védőruha visszнyereménye
041	Első kiszerelésű üzemanyag visszaadása
042	Első kiszerelésű egyéb anyag visszaadása
043	Kincstári eszköz selejtezésének visszнyereménye
044	Kincstári eszköz selejtezésének visszнyereménye
045	Vételezés javításra
046	Felhasználásra vételezés
052	Vontatójárművekhez vételezett gázolaj vontatási célra
053	Egyéb célra vételezett üzemanyag
054	Első kiszerelésként vételezett üzemanyag
055	Első kiszerelésként vételezett egyéb készlet
056	Egyen- és formaruha felhasználásra vételezése
057	Munka- és védőruha felhasználásra vételezése
058	Idegen szerv részére bérmunkára átadott készlet
122	Üdítőital vételezés

Összegzés

A program kialakítása során törekedtem áttekinthető, egyszerűen kezelhető felületeket létrehozni a felhasználó számára.

Igyekeztem a program készítése során minél több Delphi komponenst alkalmazni, azok működését bemutatni.

Az elkészült alkalmazás a kitűzött célokat megvalósítja, a vételező munkájának megkönnyítésére, gyorsabbá tételére szolgál.

A későbbiekben megvalósítható feladatok lehetnek:

> A bizonylaton lévő vonalkód segítségével egy központi feldolgozás válhat lehetővé,
> Vonalkód leolvasó rendszer kiépítése,
> A program hálózati rendszerben való alkalmassá tétele.

Köszönetnyilvánítás

Szeretnék köszönetet mondani Dr. Bajalinov Eriknek, hogy tanácsaival, javaslataival segítette a szakdolgozatom elkészítését.

Irodalomjegyzék

Marco Cantú:

Delphi 7 mesteri szinten I-II. kötet

Kuzmina Jekatyerina, Dr. Tamás Péter, Tóth Bertalan:

Programozzunk Delphi 7 rendszerben!

Baga Edit:

Delphi másképp

Michael J. Hernandez

Adatbázis-tervezés

Szabó László

Adatbázis-kezelés a Delphi segítségével

Borland Delphi 7 Professional: **Help**

www.prog.hu