Querying event logs

Jo Swinnen

promotor :
Prof. dr. Koen VANHOOF

Masterproef voorgedragen tot het bekomen van de graad van master in de toegepaste economische wetenschappen: handelsingenieur in de beleidsinformatica, afstudeerrichting informatie- en communicatietechnologie

universiteit hasselt

Woord vooraf

Deze masterproef wordt voorgedragen ter voltooiing van de opleiding Handelsingenieur in de Beleidsinformatica – ICT aan de universiteit Hasselt. Het onderwerp van deze thesis, 'Querying event logs', is een pril topic dat meer en meer op de voorgrond treedt in de huidige, real-time, economie.

Graag zou ik langs deze weg enkele personen willen bedanken die mij geholpen en gesteund hebben tijdens de verwezenlijking van deze masterproef. Eerst en vooral zou ik graag mijn promotor, prof. dr. Koen van Hoof willen bedanken voor zijn professioneel advies en opbouwende kritiek doorheen heel het verloop van mijn masterproef. Ik heb echt veel bijgeleerd door dit werk te maken. Daarnaast zou ik Els Hannes die mij heeft bijgestaan tijdens mijn praktijkstudie willen bedanken. Haar tips en ondersteuning hebben er mede voor gezorgd dat ik een nieuwe benadering op de aangereikte dataset heb kunnen voltooien. Mijn dank gaat ook uit naar Feng Liu die voor de dataset heeft gezorgd en die telkens weer klaar stond om opkomende problemen in verband met de data op te lossen.

Vervolgens zou ik graag mijn ouders willen bedanken voor de morele steun en het geduld tijdens het maken van mijn masterproef. Ik wil hen ook oprecht bedanken voor de kansen die ze mij hebben gegeven in al die jaren. Ook vrienden, familie en kennissen zou ik willen bedanken voor hun steun, aanmoediging en de nodige ontspanning.

In het bijzonder wil ik mijn vriend Jasper bedanken voor alles. Vooral voor zijn vertrouwen in mij om de overstap te maken naar beleidsinformatica twee jaar geleden. Zonder hem had ik dit nooit gedaan.

Mei 2010,
Jo Swinnen

Samenvatting

Organisaties van vandaag zijn complexer dan ooit. Heel wat processen zijn gekoppeld aan elkaar en heel wat gegevens stromen doorheen organisaties. Vroeger werden gegevens enkel opgeslagen in een relationele database zodat er daarna SQL-queries op uit konden gevoerd worden. Er zijn echter een aantal nieuwe applicaties zoals netwerk monitoring, financiële analyse, productie en sensor netwerken waarbij de gegevens niet worden opgeslagen in een relationele database maar als events in een datastroom. Een event log wordt gedefinieerd als 'alles dat gebeurt of niet gebeurt waar de business potentiële interesse in heeft'. Een event wordt gekenmerkt door een timestamp, het tijdstip waarop een bepaalde stap of activiteit heeft plaatsgevonden.

Er vinden continu nieuwe events plaats en het is moeilijk om de betekenisvolle events uit een oneindige dataset te halen. Er zitten regelmatig bepaalde patronen in deze stroom van events. Soms is een bepaald patroon in een datastroom zelfs vereist, maar sluipen er fouten in het systeem. Het komt daarnaast ook voor dat een verwacht event toch niet plaatsvindt, zogenaamde non-events. Het is belangrijk in organisaties dat de betekenisvolle events, patronen of non-events in alle datastromen real-time gedetecteerd kunnen worden.

Deze masterproef handelt over continue querytalen waarmee een gebruiker real-time op zoek kan gaan naar deze betekenisvolle events, patronen of non-events. Deze querytalen zijn gebaseerd op het algemeen gekende SQL om betekenisvolle data uit relationele databases te halen. De continue querytalen zijn dan ook herkenbaar voor een doorsnee SQL gebruiker. Er zijn natuurlijk ook heel wat nieuwe aspecten en verschillen tussen SQL en continue querytalen. Zo zijn er een aantal nieuwe functies om bijvoorbeeld patronen te detecteren en om datastromen op te splitsen in kleinere delen met sliding windows. Continue querytalen zorgen er dus voor dat het niet meer zo moeilijk is om oneindige datastromen te verwerken.

Continue querytalen zijn ontstaan in het teken van complex event processing, een event-driven architectuur waarbij 100 tot 1000 events per seconden verwerkt kunnen worden. Op deze manier kan een continue inputstroom real-time verwerkt en geanalyseerd worden om zo een continue outputstroom te verkrijgen zodat bedrijven real-time kunnen reageren op problemen en opportuniteiten die zich voordoen. Dit is onmisbaar in de real-time economie de dag van vandaag.

Er zijn daarenboven heel wat van deze CEP platformen beschikbaar, elk met hun eigen querytaal. In dit werk worden een aantal van deze platformen (StreamBase en Coral8) en continue querytalen (StreamSQL, CCL en EPL) geanalyseerd op het vlak van algemene werking, querystructuur en het gebruik van deze queries in een praktijkstudie.

De praktijkstudie wordt uitgevoerd op de dataset ‘Verplaatsingsgedrag van gezinsleden’. Deze dataset bevat gegevens van verplaatsingen van 500 gezinnen doorheen één dag tussen thuis, het werk en school. Aan de hand van vijf projecten komen een aantal belangrijke functies van continue querytalen aan bod om zo de werking en de queries te vergelijken tussen de verschillende tools. Het uiteindelijke doel is om te kijken welke querytaal het gemakkelijkst hanteerbaar en gebruiksvriendelijk is. Er wordt dus gezocht naar de meest efficiënte tool om queries uit te voeren op event logs.

Inhoudstabel

Lijst met figuren

Lijst met tabellen

HOOFDSTUK 1. Onderzoeksplan

1.1 Probleemstelling

Organisaties van vandaag zijn complexer dan ooit. Heel wat processen zijn gekoppeld aan elkaar en gegevens stromen doorheen bedrijven. Vroeger werd de meeste data opgeslagen in een relationele database. Een traditionele, relationele database bevat een reeks opgeslagen gegevens waar gemakkelijk mee gewerkt kan worden. Momenteel zijn er een aantal nieuwe applicaties zoals netwerk monitoring, financiële analyse, productie en sensor netwerken, waarbij de gegevens niet worden opgeslagen in een database maar als events in een datastroom (Arasu, et al., n.d.).

Telkens wanneer een werknemer dus een specifieke taak uitvoert, wordt deze taak in de applicatie geregistreerd als een business event log. Deze log bevat informatie over de uitvoering van het proces. Een compleet bedrijfsproces bestaat uit de verzameling logs van de verschillende taken (Segers, 2007). Doordat een bepaald proces meermaals uitgevoerd wordt op een dag door meerdere personen, lopen de verschillende taken van deze processen vaak door elkaar. Deze taken worden geregistreerd als events in chronologische volgorde waardoor er vaak geen structuur in deze datastroom zit. Hierdoor wordt het moeilijk om een antwoord te formuleren op specifieke vragen zoals hoeveel tijd heeft een werknemer nodig om taak A tot een goed einde te brengen? Of hoeveel taken werkt een werknemer af op één dag?

Event logs kunnen heel complex worden met als resultaat dat bijvoorbeeld SQL-queries, die uitgevoerd moeten worden op deze logs, ook heel complex kunnen zijn (Pieciukiewicz, Stencel, & Subieta, n.d.). Vaak is het zo dat enkel query specialisten deze queries kunnen begrijpen en uitvoeren. Er zijn ondertussen heel wat nieuwe querytalen ontwikkeld om dit probleem op te lossen. Deze querytalen, zogenaamde continue querytalen, zorgen er voor dat er wel op eenvoudige wijze vragen gesteld kunnen worden aan een datastroom. Op deze manier kunnen er ongewone patronen ontdekt worden in een proces, taken die niet hebben plaatsgevonden of kan men achterhalen welke taak te veel tijd in beslag neemt.

Er zijn dus verscheidene tools voorhanden om queries uit te voeren op event logs, maar wat is nu de beste, en/of meest efficiënte tool?

1.2 Onderzoeksvragen

1.2.1 Centrale onderzoeksvraag

Het uiteindelijke doel van deze masterproef is om een conclusie te formuleren welke tools er voor handen zijn, welke tool het efficiëntste werkt en welke tool de meeste toegevoegde waarde oplevert. Welke tool is het best hanteerbaar en welke is het gemakkelijkste uit te proberen? De centrale onderzoeksvraag luidt dan ook:

Welke tools kunnen gebruikt worden om queries uit te voeren op event logs en welke tool heeft de meeste toegevoegde waarde?

Met een grondig onderzoek tracht deze masterproef een duidelijk antwoord te formuleren en een suggestie te geven over de beste oplossing voor een doorsnee gebruiker om queries uit te voeren op event logs.

1.2.2 Deelvragen

Om een volledig antwoord te kunnen geven op de centrale onderzoeksvraag is het nodig om een aantal deelvragen te formuleren, die in de loop van het onderzoek beantwoord moeten worden.

De volgende deelvragen worden onderzocht:

A. *Wat is een event log en hoe past dit begrip binnen een organisatie?*
De tools die onderzocht worden, gaan uitgevoerd worden op event logs. Daarom is een algemene schets over het begrip 'event log' binnen een organisatie zinvol met behulp van begrippen als 'event-driven architecture' en 'event-driven system'.

B. *Wat is het verschil tussen een database management systeem(DBMS) en een datastream management systeem(DSMS)?*
Het 'oude' systeem en het nieuwe systeem worden hier vergeleken om de evolutie weer te geven naar een DSMS en om te analyseren wat de verschillen zijn tussen een DBMS en een DSMS.

C. *Wat is complex event processing en welke platformen zijn er voor handen?*
Continue querytalen zijn ontstaan in de nieuwe revolutionaire CEP-omgeving, complex event processing. Voordat deze querytalen beschreven kunnen worden, is het noodzakelijk om deze CEP-omgeving te beschrijven. Er zijn heel wat bedrijven die een CEP-platform aanbieden met elk hun eigen querytaal.

D. *Welke tools worden met elkaar vergeleken?*
Natuurlijk kunnen niet alle CEP platformen aan bod komen, vandaar dat deze masterproef zich gaat beperken tot een tweetal platformen, StreamBase en Coral8, met hun querytaal. Zowel de algemene werking als de structuur van de querytalen worden uitgelegd. Daarnaast wordt er eerst een derde querytaal, EPL, uitgelegd vermits deze een goed algemeen beeld schept van de meest voorkomende queries in continue querytalen.

E. *Wat zijn de gelijkenissen/verschillen tussen deze tools?*
Nadat de verschillende querytalen uitvoerig beschreven zijn, wordt er een vergelijking gemaakt tussen zowel de algemene werking van de CEP-platformen als hun querytalen en hun eigenschappen. Welke querytaal is het meest efficiënt?

F. *Hoe werken deze tools in de praktijk?*
Met behulp van een case wordt er getracht te ontdekken hoe de verschillende tools/querytalen in de praktijk werken. Werkt de theorie ook efficiënt in de praktijk? Hier wordt ook de vergelijkende studie tussen de verschillende tools in de praktijk verder gezet.

1.3 Dataset

Doorheen de masterproef zal telkens verwezen worden naar de case waarop de verschillende tools getest worden. De dataset 'TravelBehaviour' gaat over verplaatsingen van gezinsleden tussen hun woning, werk en school van de kinderen. Deze gegevens zijn een subset afkomstig van de socio-economische enquete van 2001 (FPS economie - Algemene directie statistiek en economische informatie, 2001). Elk individu van een gezin moest een enquête invullen over zijn/haar verplaatsingen doorheen de dag. De variabelen van een event zijn de volgende: 'HousholdID', 'PersonID', 'Activity', 'Age', 'Sex', 'HouseholdType', 'TravelActivity', 'Traveltime', 'Mode', 'ChildTransportation', 'Cars', 'Morning' en 'WorkStatus'. Deze variabelen worden verklaard in hoofdstuk 4 voordat de querytalen en de casestudie aan bod komen. Door gebruik te maken van de querytalen wordt er getracht te ontdekken of er bepaalde patronen in het verplaatsingsgedrag zitten en of er bepaalde zaken niet kloppen. Het uiteindelijke doel van heel de casestudie is om te kijken of er inderdaad efficiënt en gemakkelijk met continue querytalen betekenisvolle events uit een datastroom gehaald kunnen worden.

1.4 Onderzoeksopzet

Eerst en vooral zal de literatuurstudie verder gezet worden om meer kennis te verkrijgen over de verschillende opties die mogelijk zijn en welke tools gebruikt gaan worden tijdens het onderzoek.

Vervolgens wordt er een uiteenzetting gedaan over het onderwerp op een conceptueel niveau. De bedoeling is dat de dataset waarop de case gebaseerd is, verwerkt wordt als voorbeeld bij deze conceptuele uiteenzetting. Ten slotte worden er een aantal queries uitgevoerd op de dataset van de case om te kijken hoe het werkt en welke tool het meest efficiënt werkt.

1.5 Tijdsplan

Mijn tijdsschema ziet er als volgt uit. Eerst en vooral wil ik in de vakantie mij informeren over de verschillende tools die voor handen zijn om te gebruiken op event logs. Mijn kennis is momenteel beperkt tot de basis van SQL en die zal zeker uitgebreid moeten worden. Daarnaast zal het ook belangrijk zijn om meer feeling te krijgen met het gebruik van queries en hoe deze tools praktisch in zijn werk gaan. Tegen november zou ik graag mijn literatuurstudie afronden.

Het tweede evaluatiemoment is in februari, daarom zou ik graag tegen dan het eerste deel van mijn thesis afronden, namelijk het conceptuele aspect.

Ten slotte, gaat in het tweede semester een case opgelost worden met behulp van de geformuleerde tools zodat het uiteindelijke doel, een duidelijk antwoord op de centrale onderzoeksvraag, voor de einddatum op papier staat.

HOOFDSTUK 2. Event logs, data stream management systeem en continue queries

2.1 Event logs

Elke stap, elke activiteit van een proces in bijvoorbeeld een ERP-, workflow-, of een document managementsysteem wordt geregistreerd in de vorm van een event. Deze events worden in chronologische volgorde weergegeven en laten zien wie welke activiteit wanneer uitvoert (Goedertier, 2008). Elk event kan meerdere keren voorkomen. Deze events worden echter telkens opnieuw geregistreerd als een uniek gegeven op een uniek tijdstip. Zo een event komt overeen met een rij in een tabel van een database. Ieder geregistreerd event wordt gekenmerkt door een aantal elementen, velden, die telkens worden opgeslagen zodra dit event heeft plaatsgevonden. Deze karakteristieken komen overeen met de kolommen in een traditionele database tabel (www.streambase.com).

Tabel 1 is een vereenvoudigd voorbeeld van een event log van de case 'verplaatsingsgedrag van gezinsleden' waarbij elke rij een uniek event voorstelt, gebaseerd op wat gezinsleden in de ochtend doen in de vorm van verplaatsingen. Enkel de 'HouseholdID', 'TravelActivity', 'PersonID' en 'TravelTime' worden gebruikt in dit voorbeeld omdat deze variabelen standaard gebruikt worden in elk soort event log. Eén van de belangrijkste eigenschappen van een event log is de 'timestamp', waarmee wordt bijgehouden wanneer een bepaald event zich heeft afgespeeld (Goedertier, 2008). Deze timestamp geeft niet altijd het exacte tijdstip weer wanneer een event zich afspeelt. Dit kan ook een volgnummer zijn, zolang de timestamp maar weergeeft dat de events in chronologische volgorde worden geregistreerd (Purich, 2009a). Daarnaast wordt ook het type activiteit opgeslagen en wie de activiteit uitvoert. Op deze manier wordt elke toestandsverandering van een activiteit vastgelegd (Goedertier, 2008).

Tabel 1: Voorbeeld event log

HouseholdID	TravelActivity	PersonID	TravelTime
001	LeaveHTime	1	08:00:12
001	LeaveHTime	2	08:00:12
002	LeaveHTime	1	08:02:34
001	LeaveHTime	3	08:05:78
003	LeaveHTime	1	08:08:56
003	LeaveHTime	1	08:08:56
001	ArriveWSTime	1	08:08:89
002	LeaveHTime	2	08:12:24
002	LeaveHTime	3	08:13:31
002	LeaveHTime	4	08:13:31

003	ArriveWSTime	1	08:13:90
004	LeaveHTime	1	08:15:12
004	LeaveHTime	2	08:16:01

Dit voorbeeld toont wanneer welk gezinslid van een bepaald gezin naar school of naar het werk vertrekt en aankomt op school of het werk in de ochtend. Een voorbeeld van een event in de case is dus het vertrek van de moeder in de ochtend naar haar werk. De aankomst op haar werk is een nieuw event, dat geregistreerd wordt. Alle verplaatsingen van de gezinsleden komen terecht in een event log zoals in figuur 1. Op deze gegevens gaan uiteindelijk de continue queries uitgevoerd worden.

Met behulp van event logs kan men verschillende analyses uitvoeren. Door een performantie-analyse van het tijdstip, waarop events zich afspelen, wordt de weergave van kwantitatieve informatie over doorlooptijden, doorloopsnelheden en wachttijden mogelijk. Op basis hier van kunnen fouten of knelpunten in processen worden ontdekt (Goedertier, 2008). In de case over het verplaatsingsgedrag van gezinsleden kan men zo onverwachte patronen in de tijd ontdekken. Bijvoorbeeld: als een kind pas om 18u van school wordt gehaald, kan dit betekenen dat dit kind naar de naschoolse opvang gaat aangezien de school meestal tussen half vier en half vijf uit is.

Daarenboven kan men met behulp van een conformiteitsanalyse op event logs onderzoeken of de processen het vooropgestelde model volgen. Zo kunnen afwijkingen vastgesteld worden waardoor men bijvoorbeeld fraude op het spoor kan geraken (Goedertier, 2008). Toegepast op de data over het verplaatsingsgedrag, kan het zijn dat een ouder die normaal ‘s morgens moest gaan werken heel de dag thuis blijft. Dit kan dan wijzen op ziekte of vakantie.

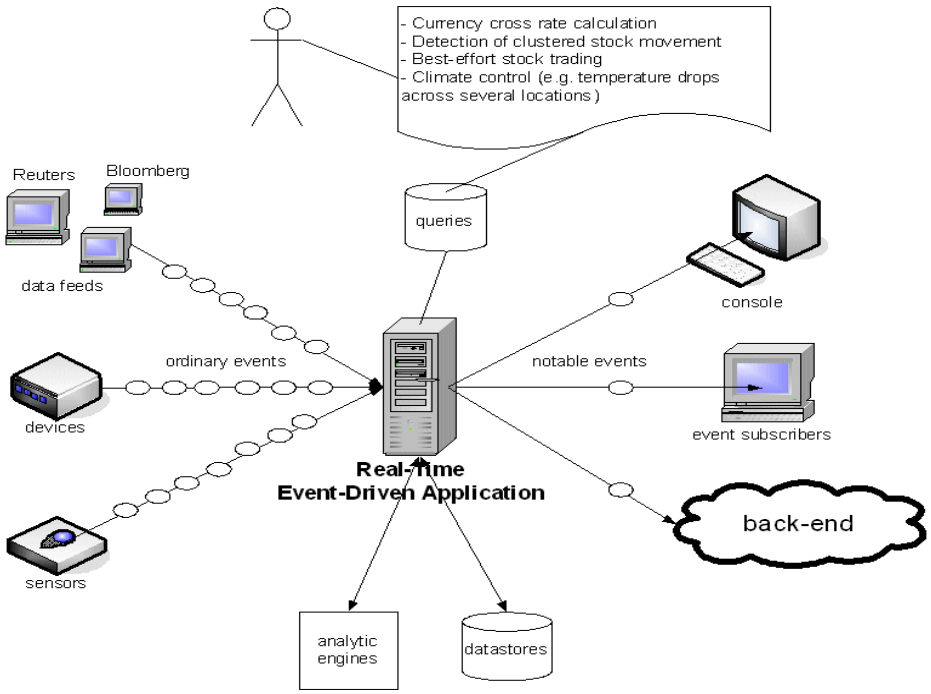
Het is vervolgens belangrijk dat men er zich van bewust is dat events niet enkel draaien rond activiteiten die plaatsvinden of veranderingen. Soms is er juist interesse in activiteiten die niet hebben plaatsgevonden, zogenaamde ‘non-events’ (Schouw, n.d. b). Het voorbeeld in de vorige alinea is hier ook van toepassing.

Schouw, n.d., verwoordt dit mooi in de volgende definitie van een event log:
‘Een event is alles dat gebeurt of niet gebeurt waar de business potentiële interesse in heeft.’

2.2 Event-driven architecture

Event-driven architecture hangt sterk samen met het begrip 'real-time'. In een real-time bedrijf wordt er onmiddellijk gereageerd indien er een verandering plaatsvindt binnen het bedrijf. Doordat deze real-time bedrijven nauw samenwerken met hun externe partners, kan worden afgeleid dat de economie zelf ook meer en meer evolueert naar een 'real-time' economie (Siegele, 2002). Dit is net hetzelfde in een event-driven architecture. Telkens wanneer een event zich voordoet in een bedrijf met een event-driven architecture, worden onmiddellijk de betrokken partijen ingelicht. Er worden daarna onmiddellijk real-time acties ondernomen (Michelson, 2006).

Figuur 1 laat een voorbeeld van een 'event-driven' systeem zien. Dit systeem bestaat meestal uit een aantal bronnen en 'sinks' van events met daar tussen de real-time event-driven toepassingen. Een datastroom van events, gecreëerd door de event bronnen, wordt onmiddellijk, real-time, verwerkt door de event-driven applicaties tot betekenisvolle gegevens. Deze gegevens worden uiteindelijk opgevangen door de 'sinks' waarin de gegevens bijvoorbeeld aangewend worden voor rapportering. De toepassingen, die de datastroom transformeren en analyseren, worden aangedreven door regels uitgedrukt in queries. De eventbronnen, de event-driven applicaties en de sinks hebben geen effect op elkaar. Dit wil zeggen dat er gemakkelijk componenten kunnen toegevoegd of verwijderd worden zonder effect te hebben op de rest van de componenten (Purich, 2009b). Complex event processing (CEP) is een belangrijke toepassing van deze event-driven architecture, zie verder in hoofdstuk 3.



Figuur 1: Event-driven systeem (Purich, 2009b)

Met behulp van continue queries wordt bijgevolg nagegaan met behulp van filter- en aggregatiefuncties welke gegevens van de inkomende datastroom zinvol zijn voor verdere analyse door externe applicaties (Purich, 2009b). Deze queries worden verder in de masterproef besproken.

2.3 DBMS versus DSMS

2.3.1 Evolutie

Gegevens van een applicatie werden altijd vroeger opgeslagen in een relationele database en omschreven als een eindige set van opgeslagen data (Babu & Widom, 2001). Traditionele databases, Data Base Management Systemen (DBMS), zijn dan ook allemaal op dezelfde manier ontworpen en bevatten een eindige reeks historische gegevens waarop gebruikers op eenvoudige wijze queries kunnen uitvoeren (Purich, 2009b).

De huidige economie is echter grondig veranderd in de tussentijd en men spreekt dan ook meer en meer van een 'real-time economy' zoals eerder vermeld. Alles gebeurt op dit moment en men probeert vertraging zo veel mogelijk te elimineren. Alle transacties en processen worden onmiddellijk geregistreerd wanneer ze uitgevoerd worden (M.A. Vasarhelyi, persoonlijke communicatie, 8 oktober,2009). Daarom bestaan de moderne databases, Data Stream Management Systemen (DSMS), van een aantal applicaties uit een stroom van data die continu verandert. Er vinden steevast nieuwe events plaats. Een dataset heeft met andere woorden geen grens meer en is dus oneindig lang (Purich, 2009b). Een datastroom bestaat uit events, geordend in chronologische volgorde. Het tijdstip waarop het event plaatsvindt, wordt telkens weergegeven (zie 2.1).

Netwerk monitoring, financiële analyse, productie- en sensornetwerken zijn een aantal voorbeelden van deze moderne applicaties die nood hebben aan queries die langdurig en continu werken (Arasu, et al., n.d.). De queries die uitgevoerd worden op een database komen hier tekort.

2.3.2 Een datastroom

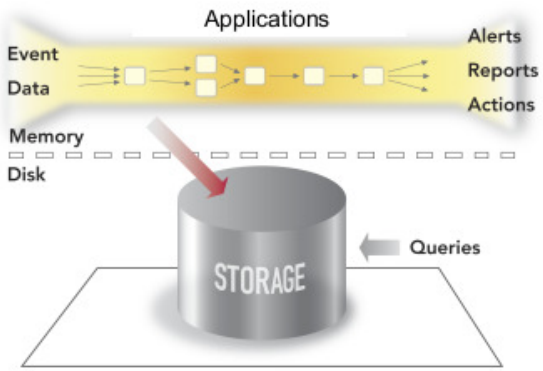
Elke datastroom heeft een aantal overeenkomstige karakteristieken waarmee rekening moet gehouden worden. Deze eigenschappen worden hier kort opgesomd (Chakravarthy & Jian, 2009).

- Events arriveren continu en sequentieel in een stroom en worden op volgorde geregistreerd in het systeem, zoals eerder vermeld, met een timestamp. Hierdoor worden events in chronologische rankschikking verwerkt door het systeem.
- Events die in een datastroom systeem binnenkomen, zijn meestal afkomstig van een externe bron of een bepaalde applicatie. Een datastroom systeem heeft dan ook geen directe toegang tot of controle over deze inputbron.
- De inputkarakteristieken van een datastroom kunnen meestal niet gecontroleerd worden en zijn vaak heel onvoorspelbaar. De snelheid waarmee deze events in het systeem binnenkomen, kan ook heel sterk variëren.
- De hoeveelheid events waaruit een datastroom bestaat, kan erg groot en zelfs onbeperkt zijn.
- Aangezien de datastroom voortkomt van een externe bron, kunnen er gemakkelijk fouten in de gegevens zitten (Chakravarthy & Jian, 2009).

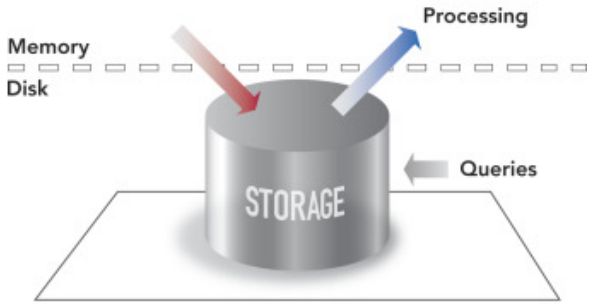
2.3.3 Vergelijking DBMS en DSMS

Figuur 2 en figuur 3 tonen het verschil tussen de werking van een database management systeem (DBMS) en een datastream management systeem (DSMS). Daarnaast geeft tabel 2 een volledig overzicht van de verschillen tussen de twee systemen.

In een datastream management systeem zoals in figuur 2 komen de gegevens het systeem real-time binnen en verlaten het systeem ook real-time. Ze worden enkel in het intern geheugen opgeslagen om ze te verwerken en er continue queries op uit te voeren zodra er gegevens binnenstromen. Bij een database management systeem in figuur 3 verloopt dit heel anders. De gegevens moeten eerst effectief opgeslagen worden, alvorens er queries op de statisch opgeslagen gegevens uitgevoerd kunnen worden (Bergamini, Biersack, Goebel, Plagemann, Tolu, & Urvoy-Keller, 2004).



Figuur 2: Datastream Management System (www.streambase.com)



Figuur 3: Database Management System (www.streambase.com)

Zowel een database als een datastroom hebben hun voordelen. Een eindige, opgeslagen dataset in een database is enerzijds belangrijk wanneer er steeds opnieuw queries uitgevoerd moeten worden op een bepaald deel van de data en wanneer er zo goed als geen updates van de data voorkomen. Anderzijds is het dynamische karakter van een oneindige datastroom belangrijk wanneer de data continu varieert of wanneer er maar enkele queries uitgevoerd moeten worden op de data (Babu & Widom, 2001) (Chakravarthy & Jian, 2009).

Een dataset in een database wordt daarenboven bevraagd met behulp van ad hoc queries terwijl bij een datastroom continue queries worden uitgevoerd. Continue queries worden ook slechts eenmaal ingevoerd op dezelfde gegevens, maar deze queries blijven zich continu herhalen op de nieuwe gegevens die binnenstromen (Babu & Widom, 2001) (Chakravarthy & Jian, 2009). Queries worden met andere woorden beschouwd als vaste entiteiten en alle nieuwe gegevens moeten door deze queries bevraagd worden (Chandrasekaran & Franklin, n.d.). Het resultaat van de queries moet bijgevolg continu geüpdatet worden telkens wanneer er nieuwe gegevens binnenkomen (Chakravarthy & Jian, 2009) (Gualtieri & Rymer, 2009).

Daarnaast is er ook een onderscheid in de resultaten. Bij een database verwacht men accurate resultaten en mogen deze resultaten geen afwijkingen vertonen. De resultaten van een datastroom mogen echter afwijkingen bevatten en benaderingen worden geaccepteerd indien er geen nauwkeurige resultaten bereikt kunnen worden (Chakravarthy & Jian, 2009). De resultaten van een query op een database worden samen weergegeven op één display als één geheel terwijl de weergegeven resultaten bij continue queries afhankelijk zijn van het gedefinieerde scherm. Dit wordt verder uitgelegd in 2.6 Sliding windows.

Tabel 2: Vergelijking DBMS – DSMS (Chakravarthy & Jian, 2009)

DBMS	DSMS
Opgeslagen, eindige dataset	Continue, oneindige datastroom
Eenmalige queries	Continue queries
Relatieve lage update frequentie	Update is belangrijk
Accurate resultaten	Benaderingen zijn toegestaan
Alle resultaten worden samen weergegeven	Resultaten zijn afh. van het scherm (zie 2.6)
Geen real-time services	Real-time vereisten

2.4 Wat is SQL?

2.4.1 Algemeen

Alle moderne relationele databases zoals Access, Microsoft SQL Server en Oracle maken gebruik van SQL. Elke continue querytaal die op events kan uitgevoerd worden, lijkt daarom ook op SQL. SQL is bijgevolg de basisquerytaal die eerst bondig uitgelegd gaat worden alvorens de complexere continue querytalen aan bod komen.

SQL, Structured Query Language, is een standaardtaal om te kunnen communiceren met, de toegang te krijgen tot en het manipuleren van relationele databases. Met behulp van SQL kunnen vragen gesteld worden aan een database in de vorm van queries om bepaalde gegevens op te vragen. Verder kan SQL er voor zorgen dat je gegevens kan toevoegen, verwijderen en updaten in een database (Chapple, n.d.).

SQL omvat twee subtalen: de data definition language (DDL) en de data manipulation language (DML). DDL wordt gebruikt om databases te creëren, te vernietigen en de structuur van tabellen aan te passen, kortom om de tabellen te beheren. Met behulp van DML kan database informatie opgevraagd, toegevoegd en gewijzigd worden. Bij routine gebruik van een database worden de commando's van DML door elke gebruiker van een database aangewend (Chapple, n.d.).

Microsoft Access is waarschijnlijk het meest gekende programma dat gebruikt wordt om een relationele database te beheren en aan te maken. Access maakt gebruik van SQL om informatie te selecteren vanuit de database met behulp van queries. Dit kan op een grafische manier gebeuren door een voorbeeld te geven van wat men wil opzoeken met behulp van 'query by example'. Zodra de query geactiveerd wordt, worden de tabellen bevraagd en wordt de nodige informatie geselecteerd. Uiteraard kunnen er ook gewoon queries ontworpen worden in SQL venster.

2.4.2 Het standaard commando

De standaard vorm van SQL bestaat uit de volgende formule:
SELECT...FROM...WHERE…

Wat onmiddellijk opvalt, is dat de woorden in hoofdletters worden geschreven. Deze conventie geldt voor alle sleutelwoorden in SQL. De niet-sleutelwoorden, zoals de naam van een tabel of een kolom, worden in kleine letters uitgedrukt. Dit commando behoort tot de data manipulation language en helpt een gebruiker om gegevens op te vragen uit een tabel. Met behulp van deze formule worden er dus gegevens geselecteerd uit bepaalde kolommen (SELECT) van een bepaalde tabel (FROM) die aan een aantal voorwaarden moeten voldoen (WHERE). Een voorbeeld zal deze formule heel wat duidelijker maken. De tabel 'leden' waarop het volgende voorbeeld gebaseerd is, is te vinden in bijlage 1.
Voorbeeld: *SELECT Naam FROM leden WHERE leeftijd>='20' AND functie = 'Admin'*

Met deze formule worden alle namen uit de tabel 'leden' geselecteerd die 20 jaar of ouder zijn (>=20) en (AND) die de functie 'Admin' uitoefenen. Het antwoord, dat deze query oplevert, is 'Anouk' en 'Philippe'.

Wanneer een tabel ontwikkeld wordt, is het belangrijk dat elke kolom een bepaald datatype krijgt toegewezen. Een query moet kunnen herkennen wat voor type gegevens in de query staan, bijvoorbeeld een getal, tekst of een tijdstip. In bijlage 2 zijn de belangrijkste, erkende datatypes van SQL te vinden.

Er zijn heel wat commando's die verwerkt kunnen worden in queries om gegevens te rangschikken, te groeperen, wiskundige formules uit te voeren op deze gegevens, ... Queries kunnen daarnaast ook andere queries bevatten, zogenaamde subqueries. Op deze manier zijn er allerlei mogelijkheden om een database te bevragen. Daarnaast zijn er ook een aantal algemeen gekende operatoren die teruggevonden kunnen worden in bijlage 3, deze worden ook gebruikt in de continue querytalen. Heel wat queries van SQL komen terug in de continue querytalen en worden daarom in hoofdstuk 5, 6 en 7 verder uitgelegd.

2.4.3 Tekortkomingen van SQL

Het probleem van SQL is dat deze taal gebaseerd is op een set eindige gegevens die vastligt. Events worden echter niet weergegeven in een set gegevens die vastligt, maar worden geregistreerd als een stroom van data. Hier komt SQL tekort en is het noodzakelijk dat deze querytaal uitgebreid wordt zodat deze toegepast kan worden op een datastroom. SQL kan daarnaast ook enkel eenmalige queries uitvoeren terwijl het bij een datastroom van events noodzakelijk is dat er continu queries uitgevoerd kunnen worden.

Traditionele databases die gebruik maken van SQL zijn niet ontworpen om data snel en continu te laden en te verwerken. Doordat de database tekort schiet, schiet SQL ook tekort en is er dus nood aan een datastroom management systeem met een nieuw verwerkingssysteem, continue queries.

Er zijn een aantal functies zoals 'join', 'sort' en 'aggregation' in SQL die niet tot een goed einde gebracht kunnen worden totdat de volledige dataset verwerkt is. Dit vormt natuurlijk een probleem door het oneindige karakter van een datastroom. Het gevolg hiervan is dat er geen output gecreëerd wordt voordat de datastroom eindigt. Voor deze 'blokkerende' functies moet daarom een oplossing gezocht worden zodat deze uiteraard niet meer voor een blokkering zorgen. De oplossing voor dit probleem is het fenomeen van een 'sliding window' (zie 2.4). Door een beperking toe te voegen aan een continue query i.v.m. het scherm kunnen deze functies wel uitgevoerd worden op een kleiner deel van de datastroom (Chakravarthy & Jian, 2009).

2.5 Continue queries

Continue queries zijn queries die voortdurend uitgevoerd worden op nieuwe gegevens om nieuwe resultaten te verkrijgen. Het zijn langdurige queries die continu nieuwe output creëren. Het zijn deze queries die geassocieerd worden met datastroom management systemen. Deze continue querytalen worden vaak gebaseerd op SQL met een aantal uitbreidingen en aanpassingen (Chakravarthy & Jian, 2009). Ze hebben dus wel veel gemeen met de queries van traditionele databases waardoor ze erg herkenbaar zijn voor een doorsnee database-gebruiker.

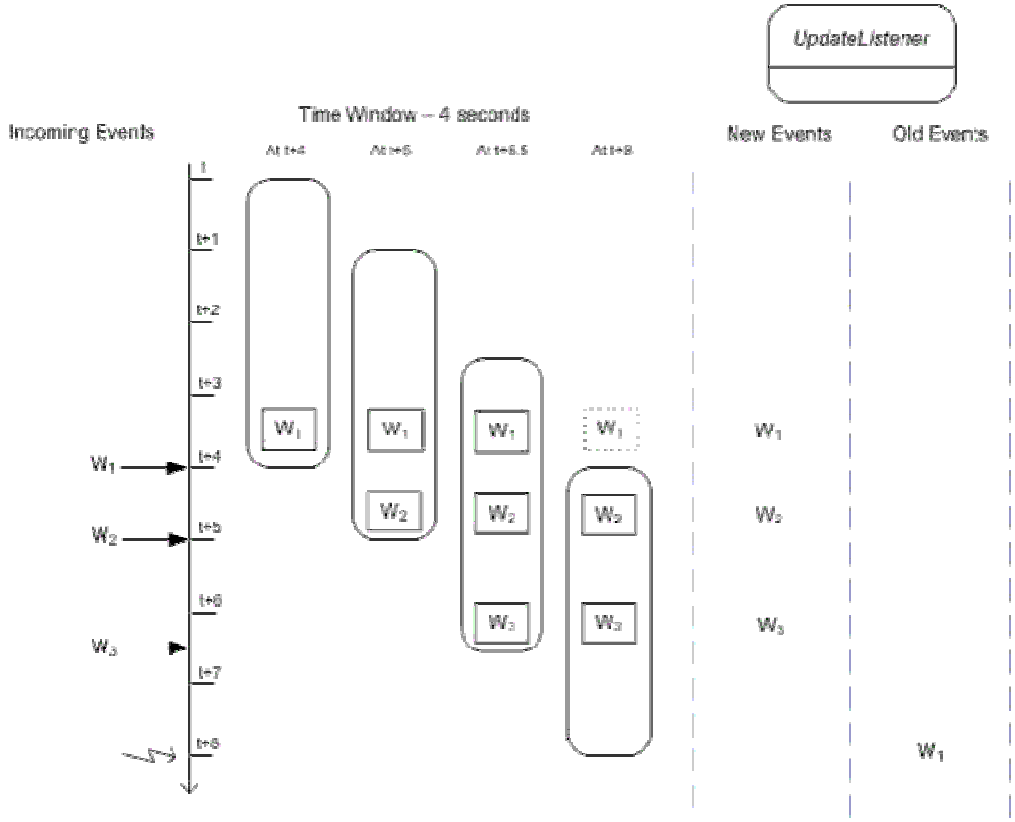
Bij een datastroom zorgen continue queries er voor dat de resultaten in de tijd geproduceerd worden en dat de stroom data, die al gepasseerd is, gereflecteerd wordt. Continue queries worden met andere woorden continu geëvalueerd doordat de datastroom constant blijft arriveren. De resultaten van deze continue queries kunnen ofwel opgeslagen en geüpdatet worden ofwel vormen deze resultaten een nieuwe stroom (Babcock, Babu, Datar, Motwani, & Widom, n.d.). Er zijn ondertussen heel wat bedrijven die een continue querytaal ontwikkeld hebben. Een aantal van deze querytalen zullen in de volgende hoofdstukken besproken worden.

2.6 Sliding windows

Een datastroom creëert constant nieuwe gegevens. Aangezien de meeste applicaties meer geïnteresseerd zijn in deze nieuwe gegevens dan in oude gegevens is het belangrijk om deze nieuwe gegevens weer te geven. Hiervoor maakt men gebruik van het begrip 'sliding window', het scherm dat de gegevens toont en waarop de queries worden uitgevoerd (Arasu & Widom,

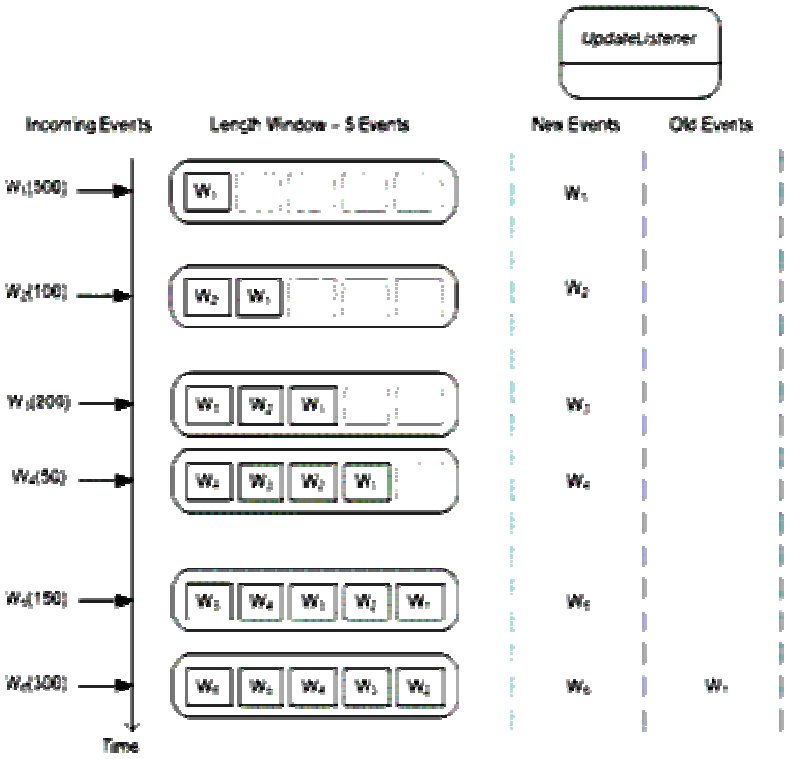
n.d.). Het gebruik van sliding windows is belangrijk wanneer de gebruiker de queries niet over heel de oneindige dataset wil uitvoeren, maar enkel op de meest recente events. Zo kan bijvoorbeeld aangegeven worden dat de query enkel mag uitgevoerd worden op de data van de voorbije week en dat data ouder dan een week verwijderd moeten worden. Op deze manier wordt er vooral nadruk gelegd op de meest recente data en dat is het belangrijkste in een real-time omgeving (Babcock, Babu, Datar, Motwani, & Widom, n.d.). De grootte van dit scherm kan bepaald worden op basis van een tijdsinterval (time-based) of op basis van een maximum aantal events die op het scherm mogen staan (row-based) (Arasu & Widom, n.d.).

Indien de grootte van het scherm afhankelijk is van een tijdsinterval, verdwijnt een 'oud' event wanneer dit interval overschreden wordt. Figuur 4 geeft hier een goed voorbeeld van. Elk event wordt in deze figuur vier seconden in het scherm weergegeven. Op tijdstip t+4 verschijnt er een nieuw event W_1 op het scherm. Na 4 seconden, op tijdstip t+8, verlaat W_1 het scherm. De gebruiker wordt telkens op de hoogte gesteld zodra een nieuw event plaatsvindt en zodra een event van het scherm verdwijnt en dus als een oud event wordt beschouwd (UpdateListener) (Purich, 2009c).



Figuur 4: Time-based sliding window (Purich, 2009c)

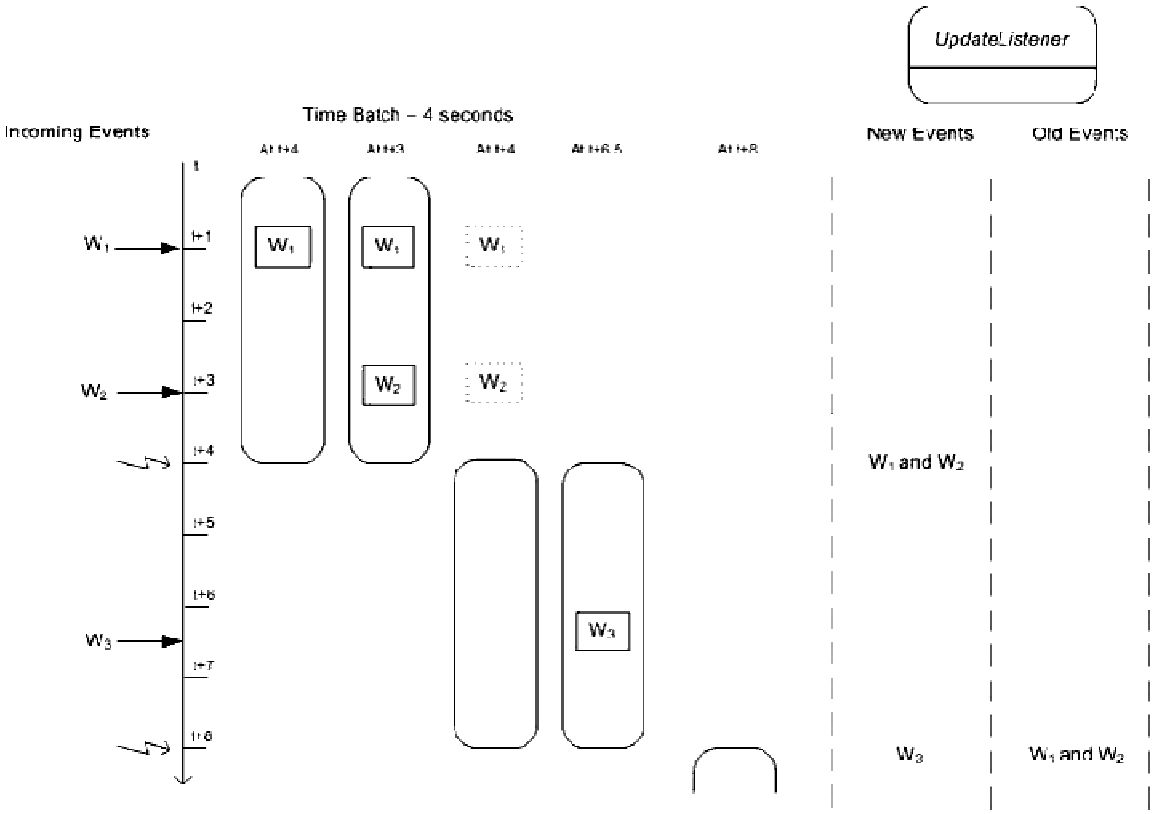
Bij de tweede optie, 'row-based sliding window', wordt er telkens gezorgd dat er maar een bepaald aantal events op het scherm staan. Telkens wanneer een nieuw event plaatsvindt en het scherm vol is, verdwijnt het oudste event van het scherm. Figuur 5 geeft een voorbeeld van een dergelijk 'row-based sliding window'. Het maximum aantal events die op het scherm te zien zijn is hier 5, dus wanneer event 6, W6, plaatsvindt, verdwijnt event 1, W1, van het scherm. De gebruiker wordt steeds weer op de hoogte gesteld van de nieuwe en oude events (Purich, 2009c).



Figuur 5: Row-based sliding window (Purich, 2009c)

De twee opties kunnen daarenboven ook op een andere manier gebruikt worden, niet met sliding windows, maar met batched windows. Op deze manier verschijnen en verdwijnen de events in groepjes op het scherm. Figuur 6 is een voorbeeld van een batched window gebaseerd op een tijdsinterval. Elk event wordt op het scherm getoond wanneer het event plaatsvindt. Zodra het gespecificeerde tijdsinterval om is, wordt de gebruiker op de hoogte gesteld van de nieuwe events (W1 en W2) die op het scherm verschenen zijn binnen het tijdsinterval. Wanneer er nogmaals dezelfde tijd verstreken is, verdwijnen de vorige events (W1 en W2) van het scherm en verschijnt het nieuwe groepje events (W3) op het scherm. De gebruiker wordt op dat moment op de hoogte gesteld dat W1 en W2 nu oude events zijn en dat W3 een nieuw event is.

Bij batches wordt de query pas uitgevoerd wanneer het scherm vol is. Nadat de query is uitgevoerd, wordt er weer gewacht tot er genoeg nieuwe events zijn om de query opnieuw uit te voeren (Purich, 2009c).



Figuur 6: Time-based batched window (Purich, 2009c)

HOOFDSTUK 3. Complex event processing platform

Er zijn heel wat softwarebedrijven die een complex event processing (CEP) platform hebben met hun eigen querytaal. De belangrijkste bedrijven, die zich met CEP bezig houden, zijn Aleri, Coral8, Oracle, Progress Apama en StreamBase. De drie querytalen die behandeld gaan worden in deze masterproef zijn StreamSQL van StreamBase, EPL (voorganger van CQL) van Oracle en ten slotte CCL van Coral8 (www.complexevents.com). EPL wordt voorgesteld om feeling te krijgen met een continue querytaal. De twee andere querytalen, StreamSQL en CCL, zijn gekozen op basis van de beschikbaarheid van informatie in de vorm van een handleiding en de toegang tot een platform om de querytalen uit te testen. In dit hoofdstuk wordt eerst CEP uitgelegd om daarna de verschillende bedrijven kort te besproken in functie van CEP. Een uitvoerige bespreking van StreamSQL, EPL en CCL komen in het volgende hoofdstukken aan bod.

3.1 CEP

CEP staat voor Complex Event Processing, bedoeld voor de ontwikkeling van event processing toepassingen. Deze toepassingen kunnen gebruikt worden om met een datastroom van 100 tot 1000 events per seconden om te gaan (McReynolds, 2007). CEP wordt gebruikt om queries uit te voeren op data alvorens deze data opgeslagen wordt in een database. Op deze manier kunnen er sneller real-time acties ondernomen worden (Schouw, n.d. a). Complex event processing is meestal zo ontworpen zodat er een aantal applicaties in geïntegreerd kunnen worden zoals Java algoritmes of wiskundige pakketten (McReynolds, 2007).

De term complex event processing is op zich eigenlijk fout. Doordat er soms 100 tot 1000 events per seconde kunnen plaatsvinden, wordt het heel moeilijk en complex om deze events te managen. Complex event processing zorgt er voor dat dit, omgaan met complexe events, vergemakkelijkt wordt en is op zich dan ook niet complex (Schouw, n.d. b).

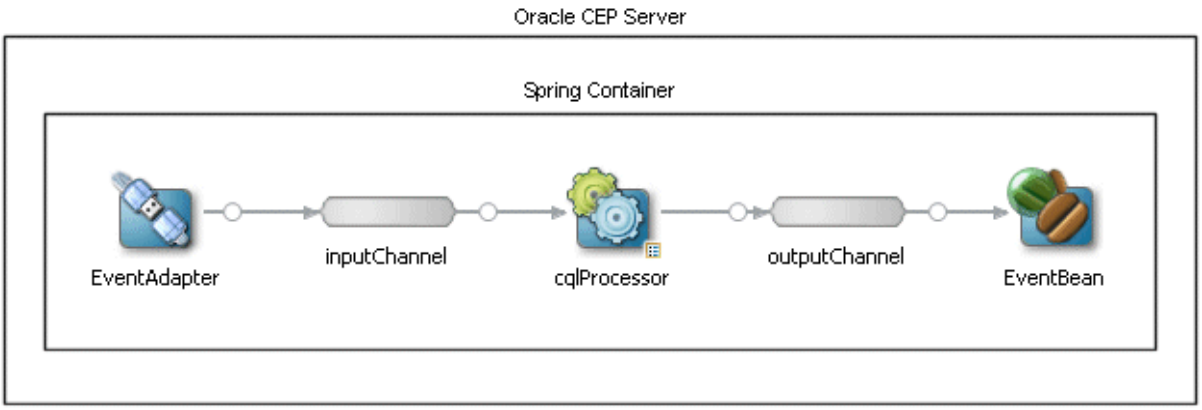
Gualtiery & Riemer (2009) geven in het Forrester Wave onderzoek de volgende definitie aan een CEP-platform:
'een CEP-platform is een software-infrastructuur die patronen van events kan ontdekken (en verwachtte events die niet hebben plaatsgevonden) door data van een live databron te filteren, te correleren, te conceptualiseren en te analyseren om onmiddellijk gepast te kunnen reageren.'

Met behulp van CEP kunnen er gemakkelijk problemen en opportuniteiten ontdekt worden doordat een CEP applicatie datastromen ontvangt, normaliseert en correleert. Tegelijkertijd

worden andere applicaties aangespoord om acties te ondernemen op basis van instructies van de CEP-applicatie. Belangrijke patronen in datastromen kunnen ook ontdekt worden met CEP. CEP zorgt voor proactieve actie zodat bedrijven zo snel mogelijk kunnen reageren op gebeurtenissen binnen het bedrijf (Schouw, n.d. a).

Een CEP-platform kan dus voor verschillende doeleinden gebruikt worden binnen de bedrijfswereld. Eerst en vooral kunnen CEP-applicaties gebruikt worden als intelligente informatiefilters in real-time applicaties, zoals een fraude detectiesysteem. Op deze manier kunnen er bepaalde relaties in continue datastromen gedetecteerd worden. Daarnaast kan een CEP-applicatie ook gebruikt worden om bepaalde patronen van events te achterhalen om hier gepast op te reageren. Ook het ontbreken van een verwacht event kan zo achterhaald worden. Als laatste kan een CEP-applicatie gebruikt worden om bedrijfsprocessen te coördineren wanneer een aantal heterogene informatiesystemen door elkaar lopen (Gualtieri & Rymer, 2009).

Een basis CEP-applicatie bestaat daarnaast uit een aantal essentiële elementen zoals bij een event-driven architecture. De CEP server van Oracle in figuur 7 wordt gebruikt als voorbeeld om een CEP-applicatie te verduidelijken. Oracle maakt gebruik van CQL, continuous query language, gebaseerd op EPL (EPL wordt in hoofdstuk 5 besproken). De eventadapter of meer bepaald de databron levert data aan een inputkanaal. Deze databron kan verschillende vormen aannemen afhankelijk van welk soort adapters door een CEP-platform ondersteund worden. Het inputkanaal of beter gezegd de inkomende datastroom is verbonden met een processor die één of meerdere queries uitvoert op de data. De queryresultaten worden vervolgens geregistreerd in het outputkanaal dat verbonden is met een 'eventbean'. Deze 'eventbean' onderneemt acties afhankelijk van de resulterende events van de uitgaande datastroom (Purich, 2009a).



Figuur 7: Oracle CEP architectuur (Purich, 2009)

Complex event processing is met andere woorden een belangrijke toepassing van een event-driven systeem (zie figuur 2). Een event-driven systeem is gebaseerd op regels geschreven in een bepaalde querytaal. Elk CEP-platform maakt gebruik van een bepaalde querytaal (bv. StreamSQL, EPL, CQL, CCL) gebaseerd op SQL. Een complex event processor moet eerst aan gegevens geraken voordat deze haar werk kan uitvoeren. Er zijn twee soorten gegevens waarmee een complex event processor kan werken. CEP is ontworpen ter ondersteuning van de combinatie van continue datastromen en opgeslagen relaties. Gegevens worden ofwel aangevoerd als een datastroom ofwel als een relatie vanuit een stroombron. Een stroombron van een datastroom kan een database zijn, een bestand of een JMS topic. De stroombron kan de gegevens naar CEP sturen, een zogenaamd push-based mechanisme of CEP kan gegevens met behulp van queries uit de stroombron halen met behulp van een pull-based mechanisme. Elke datastroom wordt gekenmerkt door twee elementen, de data zelf en een timestamp zoals eerder vermeld in 2.1 (McReynolds, 2007) (Purich, 2009).

Naast een datastroom zijn er ook nog relaties. Dit zijn verzamelingen van gerelateerde events waar geen tijdsbepaling gegeven is, net zoals bij een traditionele database. Op deze manier kunnen opgeslagen data, relaties, en datastromen gecombineerd worden. De combinatie van relaties en datastromen is één van de sterkste punten van een CEP-platform. Het schema dat voor elk event wordt vastgelegd in een datastroom, moet ook toegepast worden bij relaties (McReynolds, 2007) (Purich, 2009).

Er kunnen heel wat verschillende functies geïntegreerd worden in de querytalen van de CEP-platformen, waaronder SQL-functies, wiskundige en statistische functies. Daarnaast kan er ook gebruik gemaakt worden van plug-in software waardoor het mogelijk wordt vooraf gedefinieerde algoritmes te gebruiken in CEP. Naast de reeds vernoemde elementen zoals stromen, relaties en functies zijn patronen ook een essentieel begrip. CEP kan patronen real-time ontdekken in datastromen en relaties. Deze begrippen zijn essentieel om de werking van CEP-platformen en de bijhorende querytalen te begrijpen (McReynolds, 2007).

3.2 De platformen

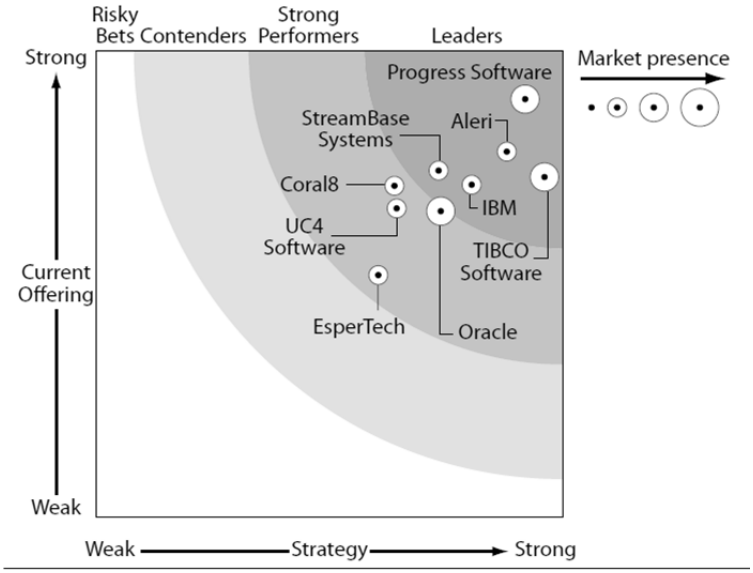
Er zijn heel wat verschillende CEP platformen. In 3.1 werd het voorbeeld van Oracle reeds aangehaald. Oracle maakt gebruik van CQL, gebaseerd op EPL (event processing language). Enkel EPL wordt in deze thesis besproken als introductie op continue querytalen (hoofdstuk 4). Naast Oracle zijn Aleri, Coral8, Progress Apama en StreamBase met hun platformen de bekendste leiders in de wereld van complex event processing. Zij zijn dan ook de stichters van de eerste CEP platformen.

Het StreamBase event processing platform is een software speciaal ontworpen door StreamBase Systems in 2003 om real-time datastromen te verwerken en te analyseren om onmiddellijk beslissingen te kunnen nemen. In dit platform worden drie belangrijke aspecten samen geïntegreerd: een grafische eventflow taal, een event server en de verbinding tussen real-time en historische data. De querytaal die door StreamBase aangewend wordt is StreamSQL (www.streambase.com). StreamBase is op de voorgrond gekomen als één van de globale technologische pioniers van 2010 op het wereld economisch forum. Er wordt dus verwacht dat StreamBase een grote impact gaat hebben op de toekomstige economie en maatschappij (Jones & W, 2009). Dit is een bijkomende reden waarom StreamBase en StreamSQL uitgebreid aan bod komen in hoofdstuk 6.

Naast StreamBase is Progress Apama één van de leiders op het vlak van complex event processing. Progress Apama is één van de eerste bedrijven die zich bezig hield met complex event processing. Het event processing platform maakt gebruik van de querytaal Apama EPL. De monitoring, het detecteren en het analyseren van patronen, real-time acties ondernemen en het correleren van meerdere datastromen zijn de belangrijkste kenmerken van dit platform. In 2009 werd Progress Apama verkozen tot leider in CEP (www.web.progress.com). Aangezien er geen directe informatie te vinden is over Apama EPL en het platform niet zo maar te downloaden is als trial, wordt dit platform verder niet meer besproken.

Aleri en Coral8 zijn ten slotte de laatste twee platformen. Deze twee worden samen vermeld vermits ze sinds 2009 gefusioneerd zijn en nu een sterke concurrent zijn op het vlak van complex event processing. Zowel Aleri als Coral8 hebben hun eigen SQL-gebaseerde querytaal. Coral8 maakt gebruik van CCL, continuous computation language, dat verder aan bod komt in hoofdstuk 7. De querytaal van Aleri komt in deze masterproef niet aan bod. Ondertussen zijn Aleri en Coral8 in februari 2010 in zee gegaan met Sybase waardoor de eventgeoriënteerde platformen van Aleri en Coral8 gecombineerd worden met de eventgeoriënteerde database van Sybase. Op deze manier proberen Aleri-Coral8-Sybase hun concurrentiepositie te vergroten (Palmer, 2010). StreamBase heeft onmiddellijk gereageerd door de klanten van Aleri-Coral8-Sybase een gratis overstap naar StreamBase inclusief een ondersteunende workshop aan te bieden (StreamBase, 2010). Er wordt dus flink gestreden voor een goede concurrentiepositie op de CEP-markt.

Forrester Wave heeft een onderzoek verricht naar de verschillende CEP platformen en hun werking. Uit dit onderzoek blijkt dat Progress Apama en Aleri de twee grootste leiders zijn in CEP. Dit onderzoek is echter gebeurd voordat Aleri en Coral8 gefusioneerd zijn en hier is dus ook geen rekening mee gehouden, staat te lezen in het onderzoek. In figuur 8 ziet u waar elk bedrijf geplaatst staat ten opzichte van elkaar in functie van hun strategie en hun aanbod (Gualtieri & Rymer, 2009).



Figuur 8:CEP-platformen en hun concurrentiepositie (www.web.progress.com)

HOOFDSTUK 4. De dataset

Alvorens de verschillende querytalen worden uitgelegd en vergeleken, wordt eerst de dataset verduidelijkt zodat deze gebruikt kan worden in de voorbeelden van de queries en in de casestudie van hoofdstuk 8.

De dataset, waarmee gewerkt gaat worden, is een subset van de gegevens van de socio-economische enquête van 2001, SEE 2001 (FPS economie - Algemene directie statistiek en economische informatie, 2001) . De dataset 'verplaatsingsgedrag van gezinsleden' omvat events van het verplaatsingsgedrag van 500 gezinnen op 1 dag. Deze subset bevat enkel gezinnen met kinderen die naar de lagere school gaan. De volgende variabelen worden telkens geregistreerd per event:

- 'HouseholdID': ID dat elk huishouden uniek maakt
- 'PersonID': ID van elk gezinslid per huishouden
- 'Activity': werk voor een ouder of school voor een kind
- 'LeaveHTime': het tijdstip waarop een gezinslid naar het werk of naar school vertrekt in minuten
- 'ArriveWSTime': het tijdstip waarop een gezinslid aankomt op het werk of op school in minuten
- 'LeaveWSTime': het tijdstip waarop een gezinslid terug vertrekt vanop het werk of school in minuten
- 'ArriveHTime': het tijdstip waarop een gezinslid thuis arriveert na het werk of school in minuten
- 'Mode': transportwijze
  (1 = te voet, 2 = fiets, 3 = brommer/motor, 4 = autobestuurder, 5 = autopassagier, 6 = bus, tram, metro, 7 = trein, 8 = collectief vervoer georganiseerd door het werk of school)
- 'BringToSchool': variabele die weergeeft welke ouder een kind in de ochtend naar school brengt (1 = ja, 0 = nee)
- 'GetFromSchool': variabele die weergeeft welke ouder een kind 's avonds van school haalt
  (1 = ja, 0 = nee)

De oorspronkelijke dataset is echter beperkt, dus een aantal uitbreidingen en aanpassingen zijn op hun plaats. Het uiteindelijk doel is om de queries uit te proberen en te vergelijken, dus de dataset wordt zo aangepast zodat er optimaal met de queries gewerkt kan worden.

Om de kolommen 'BringToSchool' en 'GetToSchool' goed te kunnen gebruiken in de queries, wordt er eerst en vooral een nieuwe kolom aangemaakt 'ChildTransporation'. Hierin wordt weergegeven welke ouder verantwoordelijk is voor het halen en/of brengen van de kinderen naar school. Deze nieuwe variabele vervangt de twee oude variabelen en kan de volgende waarde aannemen:
1: de ouder die de kinderen enkel naar school brengt
2: de ouder die de kinderen enkel van school haalt
3: de ouder die de kinderen zowel brengt als haalt van school
4: de ouder die de kinderen niet brengt en haalt van school
5: als het over het kind zelf gaat

Om nog meer queries te kunnen uittesten, worden er daarenboven nog een aantal extra variabelen toegevoegd. De leeftijd van elk individu, 'PersonAge', kan belangrijk zijn om een onderscheid te maken tussen kinderen onder 12 jaar en kinderen boven 12 jaar. Een kind boven 12 jaar kan normaal zelfstandig naar school gaan terwijl kinderen onder 12 jaar begeleiding nodig hebben van een volwassene (Hannes, et al., 2010). Deze dataset gaat enkel over kinderen die nog naar de lagere school gaan, dus hier kan onderzocht worden of deze evolutie al op jongere leeftijd zichtbaar is.

Vervolgens wordt ook het geslacht, 'Sex', van elk persoon toegevoegd (man = 1, vrouw = 2). Deze variabele kan interessante resultaten opleveren vermits het meermaals voorkomt dat vrouwen eerder zorgen voor het transport van kinderen dan mannen (Hannes, et al., 2010). Daarnaast wordt een onderscheid gemaakt tussen de verschillende huishoudtypes met de variabele 'HouseholdType': eenoudergezinnen (1), tweeoudergezinnen (2) waarbij beide ouders werken en tweeoudergezinnen waarbij een ouder werkt (3).

De volgende drie variabelen worden ook nog toegevoegd om meer mogelijkheden te bieden in de casestudie:

- 'Cars': aantal auto's in een gezin
- 'Morning': variabele die aangeeft of een kind vanaf thuis of vanaf de kinderopvang naar school vertrekt in de ochtend
  (Y = vanaf thuis naar school, N = vanaf de kinderopvang naar school, Z = onbekend)
- 'WorkStatus': variabele die weergeeft wat de werkstatus is van een gezinslid
  (0 = werkloos of student, 1= parttime, 2 = fulltime, 9999 = onbekend)

De dataset bestaat nu uit 15 variabelen, namelijk: 'HouseholdID', 'PersonID', 'Activity', 'PersonAge', 'Sex', 'HouseholdType', 'LeaveHTime', 'ArriveWSTime', 'LeaveWSTime', 'ArriveHTime', 'Mode', 'ChildTransportation', 'Cars', 'Morning' en 'WorkStatus'. Deze dataset gaat

voor bepaalde projecten gebruikt worden waarbij het huidige tijdstip als timestamp gebruikt wordt, maar voor een aantal projecten is het gebruik van een andere dataset eenvoudiger.

Het is immers belangrijk dat de events in chronologische volgorde worden weergegeven. Dit is momenteel niet zo aangezien de eigenlijke events per gezinslid naast elkaar staan: 'LeaveHTime', 'ArriveWSTime', 'LeaveWSTime' en 'ArriveHTime'. De chronologische volgorde kan gerealiseerd worden door de namen van de vier variabelen die de soort verplaatsing voorstellen, 'LeaveHTime', 'ArriveWSTime', 'LeaveWSTime' en 'ArriveHTime', samen te voegen in een nieuwe variabele 'TravelActivity'. Daarnaast wordt er een tweede nieuwe variabele 'TravelTime' gecreëerd die het tijdstip van elke soort verplaatsing vastlegt. In Excel kan een dataset vervolgens heel eenvoudig gesorteerd worden op basis van een bepaalde kolom, hier de kolom van variabele 'TravelTime'. Op deze manier worden de events in chronologische volgorde geplaatst.

Het tijdstip van elk event staat echter nog niet in het juiste formaat opdat deze gebruikt kan worden als datatype 'Timestamp'. Het juiste formaat van de data is afhankelijk van het platform, dit wordt later verder uitgelegd. In de oorspronkelijke dataset staat het tijdstip uitgedrukt in minuten. Dit zou natuurlijk aangepast kunnen worden naar het juiste formaat, maar er zijn te veel events die op hetzelfde aantal minuten plaatsvinden, waarvan het aantal seconden niet geweten is, waardoor het moeilijk wordt om te bepalen welk event voorrang krijgt op een ander event. Toch is er voor een deel van de casestudie het formaat aangepast en is er gekozen voor een willekeurige volgorde van de events met hetzelfde tijdstip. Er wordt steeds vermeld met welke dataset gewerkt wordt.

HOOFDSTUK 5. Event processing language

5.1 Algemeen

Event Processing Language of EPL is een continue querytaal ontworpen door Oracle. Het is de voorloper van Complex Query Language of kortweg CQL. Deze querytaal is speciaal ontworpen voor CEP om queries te kunnen uitvoeren op datastromen. EPL wordt in deze masterproef beschouwd als een kennismaking met continue queries. In hoofdstuk 6 en 7 komen StreamSQL en CCL dan uitgebreider aan bod in combinatie met hun CEP-platform om daarna deze platformen en queries te testen en te vergelijken in hoofdstuk 8.

Het EPL procesmodel is continu. Er zijn twee soorten events, nieuwe events die het outputscherm binnen komen en oude events die het outputscherm verlaten. Dit gebeurt afhankelijk van de instellingen van de 'sliding windows' (zie 2.6). Telkens wanneer er nieuwe events voldoen aan de beperkingen die zijn opgelegd, wordt er onmiddellijk output gecreëerd. Het is met andere woorden een real-time proces (Purich, 2009c).

EPL lijk op SQL met commando's als 'SELECT', 'FROM', 'WHERE', 'GROUP BY', 'HAVING' en 'ORDER BY'. Het verschil tussen EPL en SQL is dat bij EPL de gegevens uit een continue stroom komen terwijl bij SQL de gegevens in een relationele tabel staan. De rijen in een tabel bij SQL komen dan ook overeen met de events bij EPL. EPL vult SQL verder aan met een aantal commando's noodzakelijk om event stromen te kunnen verwerken.

Indien een bepaald event aan de voorwaarden van een EPL-query voldoet, wordt de event-sink hiervan op de hoogte gebracht. Nieuwe events die voldoen aan de EPL-query, worden ISTREAM (insert stream) events genoemd. Oude events die voldeden aan de EPL-query worden na een tijd uit het scherm verwijderd en worden RSTREAM (removing stream) events genoemd. Het binnenkomen van nieuwe events en het verwijderen van oude events is afhankelijk van de sliding windows (zie 2.6) (Purich, 2009c). Dit wordt later in 5.5 verder toegelicht.

5.2 Conventies, gereserveerde woorden en datatypes

Er zijn een aantal belangrijke conventies die men toepast in EPL. Deze moeten gekend zijn om de verschillende commando's te kunnen begrijpen en te ontwerpen.

- Sleutelwoorden worden steeds in HOOFDLETTERS weergegeven. Dit zijn gereserveerde woorden die niet voor een ander doeleinde gebruikt mogen worden.

- Vervangbare items worden in *kleine, schuin gedrukte letters* weergegeven.
- [Optionele items worden weergegeven tussen haakjes]
- Een keuze wordt uitgedrukt met een schuine streep / tussen 2 items die tussen {accolades} staan
- Indien er meerdere optionele inhoud mogelijk is, wordt dit aangeduid door [, ...]
- Na elke query volgt een ; als afsluiting van die query;
- Wildcards: * wordt gebruikt om alle velden samen te selecteren.

De datatypes zijn hier niet van belang vermits het definiëren van de data in gebeurt in Java in plaats van in EPL. De datatypes moeten wel gerespecteerd worden in de queries.

5.3 Commando's

EPL heeft enkel DML-commando's en geen DDL-commando's. StreamSQL en CCL hebben dit wel. Enkel de belangrijkste commando's aan bod (Purich, 2009c).

Een standaard EPL-query die uitgevoerd kan worden op een event kan uit volgende commando's bestaan:

[INSERT INTO *insert_into_def*]
SELECT *select_list*
{ FROM *stream_source_list* / MATCHING *pattern_expression* }
[WHERE *search_conditions*]
[GROUP BY *grouping_expression_list*]
[HAVING *grouping_search_conditions*]
[ORDER BY *order_by_expression_list*]
[OUTPUT *output_specification*];

De verschillende opties van deze query worden nu afzonderlijk aangehaald om de verschillende commando's te verduidelijken.

SELECT [ISTREAM| RSTREAM] *select_list (*AS *new_name)* FROM *stream_source_list* RETAIN;
Het SELECT-commando is vereist in alle EPL-queries en bepaalt welke velden, *select_list*, van de events in de resulterende, uitgaande datastroom terecht komen. Indien alle velden gewenst zijn, wordt, zoals bij SQL, een asterisk * gebruikt. Indien er specifieke velden gevraagd worden, worden deze bij hun naam genoemd. Er kunnen ook bewerkingen van meerdere velden aangemaakt worden zoals een vermenigvuldiging van twee velden. Om meer duidelijkheid te krijgen, als er bijvoorbeeld gebruik wordt gemaakt van bewerkingen, kan een nieuwe naam gegeven worden aan de volledige bewerking met behulp van 'AS' (Purich, 2009c).

Voorbeeld: SELECT *
SELECT HouseholdID, PersonID
SELECT (ArriveWSTime-LeaveHTime) As TravelMinutes

Daarnaast kan er ook duidelijk gemaakt worden of er enkel nieuwe of oude events getoond worden door gebruik te maken van 'ISTREAM' en 'RSTREAM.' 'SELECT ISTREAM' betekent dat enkel de inkomende events getoond worden, terwijl 'SELECT RSTREAM' betekent dat enkel de uitgaande events gepresenteerd worden. Wanneer dit niet vermeld wordt, worden zowel de nieuwe als oude events bekendgemaakt aan de gebruiker (Purich, 2009c).

Het 'SELECT'-commando komt daarenboven meestal voor in combinatie met het 'FROM'-commando. Dit commando geeft weer van welke stroombron de data afkomstig is, '*stream_source_list'*, dus wat de inkomende datastroom is. Dit deel van de query kan ook een subquery bevatten waarmee eventstromen samengevoegd kunnen worden of waarmee de juiste structuur aan de eventstroom gegeven kan worden in de 'hoofd' query. Deze subquery kan ook gebruikt worden om de inkomende datastroom te filteren alvorens er een query op uitgevoerd wordt. (Purich, 2009c) Dit wordt verder besproken in 5.4.
Hier volgen twee voorbeelden. In het eerste voorbeeld worden alle velden van alle events van de inputstroom 'TravelBehaviour' geselecteerd. Het tweede voorbeeld selecteert de 'HouseholdID' en de 'PersonID' van alle events uit de inkomende stroom 'TravelBehaviour' waarbij de wijze van transport, mode, gelijk is aan vier (bestuurder van de auto). De events die voldoen aan deze query, worden telkens voor tien seconden in het scherm weergegeven. Dit voorbeeld wordt verder verduidelijkt bij de uitleg van de volgende commando's.

Voorbeeld: SELECT * FROM TravelBehaviour RETAIN ALLE EVENTS;
SELECT HouseholdID, PersonID FROM (SELECT * FROM TravelBehaviour WHERE Mode = 4) RETAIN 10 SECONDS;

Zoals te zien in de voorbeelden wordt achter de stroombron bepaald hoeveel events er gelijktijdig als output weergegeven worden met behulp van 'RETAIN'. 'RETAIN' definieert eigenlijk het scherm van events waarop de query uitgevoerd zal worden voor elke stroombron. Indien er meerdere stroombronnen zijn, kan er per stroombron het aantal events bepaald worden (Purich, 2009c). Het volgende voorbeeld geeft dit weer door te veronderstellen dat er twee inkomende stromen zijn van het verplaatsingsgedrag. Enkel nieuwe events worden weergegeven.

Voorbeeld: SELECT ISTREAM * FROM TravelBehaviour1 RETAIN 20 SECONDS, TravelBehaviour2 RETAIN 15 SECONDS;

Dit moet echter niet, het kan ook zijn dat het aantal events gelijk is voor alle stroombronnen waardoor één 'RETAIN' voldoende is. Het FROM-commando gaat steeds gepaard met 'RETAIN'.

Er is één uitzondering: indien er een subquery aanwezig is, wordt 'RETAIN' niet gebruikt. Het is belangrijk om te weten dat de query enkel wordt uitgevoerd op de events in het scherm. Zodra er nieuwe events arriveren of oude events weggaan, wordt de query opnieuw uitgevoerd (Purich, 2009c). Meer toepassingen van 'RETAIN' zijn te vinden in 5.5 waar de toepassing van sliding windows bij EPL wordt uitgelegd.

SELECT [ISTREAM│RSTREAM] *select_list* MATCHING *pattern_expression*;
Naast het FROM-commando, kan 'SELECT' ook gecombineerd worden met 'MATCHING'. Met behulp van dit commando kunnen events geselecteerd worden die voldoen aan een bepaald patroon. Zo kunnen er patronen van opeenvolgende events ontdekt worden met 'FOLLOWED BY'. Op deze manier kan een bepaalde ordening in events naar boven komen. Ook de afwezigheid van bepaalde events kan opgemerkt worden met de 'NOT'-operator. De 'AND'- en de 'OR'-operator kunnen gebruikt worden om een combinatie van bepaalde events te onderzoeken en hoe deze combinatie voorkomt. Nadat een bepaald patroon ontdekt is, kan met 'EVERY' gekeken worden of dit patroon zich blijft herhalen. Met 'WITHIN' kan er ook een tijdsinterval bepaald worden waarin het patroon zich moet afspelen. Er zijn dus heel wat mogelijkheden om betekenisvolle patronen aan het licht te brengen (Purich, 2009c).

Voorbeeld: SELECT * MATCHING (A FOLLOWED BY B) AND NOT C WITHIN 10 SECONDS;

De rest van de commando's zijn optioneel bij EPL, maar komen wel regelmatig voor.

SELECT *select-list* FROM *stream_source_list* WHERE *search_conditions*;
Opdat de inkomende events aan één of meerdere voorwaarden zouden voldoen, kan er ook gebruik gemaakt worden van 'WHERE'. Met behulp van dit commando kunnen eventstromen samengevoegd of gefilterd worden. Indien er meerdere voorwaarden gebruikt worden, kunnen 'AND', 'OR' en 'NOT' helpen om deze voorwaarden te combineren. Net als bij SQL is er een belangrijk verschil met het commando 'HAVING' (zie verder). De operators die ondersteund worden door 'WHERE' zijn: =, <, >, >=, <=, !=, <>, 'IS NULL', 'IS NOT NULL', 'AND' en 'OR'. Aggregatie functies kunnen enkel in combinatie met het commando 'HAVING' voorkomen en mogen hier dus niet gebruikt worden (Purich, 2009c). Twee voorbeelden gaan het gebruik van 'WHERE' verduidelijken. Het eerste voorbeeld gaat events selecteren van de inputstroom 'TravelBehaviour' waarbij 'Activity' gelijk is aan 'S' (School) en de 'Mode' gelijk aan 5 (Passagier in de auto). Op deze manier worden alle verplaatsingen van kinderen als passagier in een auto geselecteerd. Het tweede voorbeeld geeft alle events weer waarbij een individu het huis verlaat (LeaveHTime) of terug thuis komt (ArriveHTime).

Voorbeeld: SELECT * FROM TravelBehaviour WHERE Activity = 'S' AND Mode = 5;

```
SELECT * FROM TravelBehaviour WHERE TravelActivity = 'LeavHTime' OR
TravelActivity = 'ArriveHTime';
```

SELECT *select_list* FROM *stream_source_list* GROUP BY *grouping_expression_list* HAVING *grouping_search_conditions*;

'GROUP BY' splitst daarnaast de resulterende events van de query in groepen op basis van één of meerdere event eigenschappen. 'GROUP BY' wordt vaak in combinatie met een 'SELECT'-commando met aggregatie functies in gebruikt. Zo worden de berekeningen per subgroep gedaan om een beter overzicht te krijgen. Het is echter niet de bedoeling dat het 'GROUP BY'-commando zelf een aggregatie functie bevat. Eigenschappen van een event die in een aggregatie functie gebruikt worden in het 'SELECT'-commando mogen ook niet terug opnieuw gebruikt worden in het 'GROUP BY'-commando. Het verschil met SQL is dat de eigenschappen die in het 'GROUP BY'-commando staan, niet noodzakelijk in het 'SELECT'-commando moeten voorkomen (Purich, 2009c). Met het volgende voorbeeld worden alle events van de inputstroom 'TravelBehaviour' van individuen die fulltime werken (WorkStatus = 2) geselecteerd, gegroepeerd per transportwijze (Mode).

Voorbeeld: SELECT * FROM TravelBehaviour WHERE WorkStatus = 2 GROUP BY Mode;

Indien 'HAVING' daarenboven gebruikt wordt in combinatie met 'GROUP BY', worden er één of meerdere voorwaarden aan de groepen events gesteld met behulp van aggregatie functies om events te elimineren uit deze groepen. Het werkt dus, net als 'WHERE' bij 'SELECT', als een filter op 'GROUP BY'. Het enige verschil is het gebruik van de aggregatie functies. Wanneer er meerdere voorwaarden gecombineerd worden, kan er weer gebruik gemaakt worden van 'AND', 'OR' en 'NOT' (Purich, 2009c). Het volgende voorbeeld gaat events groeperen naargelang hun 'HouseholdID' waarbij de 'gemiddeldeReisTijd' in de ochtend meer dan 20 minuten bedraagt.

```
Voorbeeld: SELECT HouseholdID, PersonID, AVG(ArriveWSTime – LeaveHTime) AS
           GemiddeldeReisTijd FROM TravelBehaviour GROUP BY HouseholdID
           HAVING GemiddeldeReisTijd > 20;
```

SELECT *select_list* FROM *stream_source_list* ORDER BY *order_by_expression_list;*

Om de resulterende events te ordenen op basis van een bepaalde eigenschap, kan het 'ORDER BY'-commando gebruikt worden. Indien er een aggregatie functie gebruikt wordt om gegevens te rangschikken, moet er voor gezorgd worden dat deze functie ook in het 'SELECT'-commando voor komt. Met behulp van 'ASC' en 'DESC' kan er, op dezelfde manier als bij SQL, bepaald worden in welke volgorde de gegevens geordend moeten worden (Punich, 2009c). In plaats van

de events weer te geven op chronologische volgorde worden de events van de stroom 'TravelBehaviour' in het onderstaande voorbeeld weergegeven, gesorteerd op 'HouseholdID'.

Voorbeeld: SELECT * FROM TravelBehaviour RETAIN 5 EVENTS ORDER BY HouseholdID;

SELECT *select_list* FROM *stream_source_list* OUTPUT *output_specification;*
Het 'OUTPUT'-commando komt vervolgens niet voor bij SQL omdat het een speciaal commando voor datastromen is. De snelheid waarmee events verwerkt worden tot output, wordt hiermee gestabiliseerd en gecontroleerd. Dit kan op basis van tijd of op basis van het aantal events gebeuren (Purich, 2009c). Hier volgt een voorbeeld waarbij de events telkens na een tijdsinterval van 20 seconden worden weergegeven.

Voorbeeld: SELECT * FROM TravelBehaviour RETAIN 5 EVENTS OUTPUT EVERY 20 SECONDS;

INSERT [ISTREAM| RSTREAM] INTO *event_type_alias (property_name[,property_name[,…]]);*
De resultaten van een bepaalde query kunnen ten slotte naar een bepaalde outputstroom gestuurd worden of ook terug opnieuw gebruikt worden als een inputstroom om hier queries op uit te voeren. De resultaten worden in deze nieuwe datastroom geplaatst met behulp van 'INSERT INTO'. De *'event_type_alias'* geeft een naam aan deze nieuwe stroom. De eigenschappen van de datastroom kunnen expliciet geformuleerd worden tussen haakjes achter de datastroom. De naam van de nieuwe stroom kan dan verder gebruikt worden in nieuwe queries. Met behulp van 'ISTREAM' en 'RSTREAM' kan gespecifieerd worden of enkel de nieuwe of enkel de oude events in deze nieuwe inputstroom geplaatst worden (Purich, 2009c).

Voorbeeld: INSERT ISTREAM INTO Outputstroom1;

5.4 Subqueries

Subqueries zijn simpelweg queries binnen een andere query. Dit kan op twee plaatsen voorkomen in een query: het 'SELECT'- en het 'WHERE'-commando. Het gebruik van subqueries kan een oplossing bieden voor complexe samenvoegingen van twee queries. Daarnaast kan een subquery ook gebruikt worden als filter zoals eerder aangehaald. Een subquery kan enkel uit de volgende commando's bestaan: 'SELECT', 'FROM' (met 'RETAIN') en 'WHERE'. Vervolgens kunnen er ook geen aggregatie functies in een subquery voorkomen, 'INSERT INTO' kan hier een oplossing voor bieden. Het is belangrijk om te weten dat indien subqueries gebruikt worden, deze steeds eerst worden uitgevoerd voordat de eigenlijke query wordt uitgevoerd (Purich, 2009c). Het volgende voorbeeld geeft een subquery weer binnen het FROM-commando.

Voorbeeld: SELECT HouseholdID, PersonID, Age FROM (SELECT * FROM TravelBehaviour
WHERE HouseholdType = 2);

5.5 Toepassing sliding windows

Het EPL procesmodel is continu. De gebruiker krijgt output zodra de events met behulp van queries verwerkt zijn. Hoeveel output er op het scherm verschijnt, wordt vastgelegd in de query. Hier komt het begrip 'sliding windows' op de voorgrond. Het scherm kan aangepast worden op basis van het aantal events, een bepaalde tijd of door gebruik te maken van batches (Purich, 2009c). Het scherm wordt bepaald met behulp van het 'RETAIN'-commando. De verschillende mogelijkheden worden uitgelegd aan de hand van een voorbeeld.

SELECT ISTREAM * FROM TravelBehaviour RETAIN ALL;
In dit voorbeeld wordt geen beperking op het scherm toegepast en blijven alle inkomende events behouden. Deze query zorgt er voor dat er enkel nieuwe events op het scherm verschijnen en dat de gebruiker niet op de hoogte wordt gesteld van oude events (Purich, 2009c).

SELECT * FROM TravelBehaviour RETAIN 5 events;
Dit is een voorbeeld van een row-based sliding window. Het scherm is zo ingesteld dat er maximaal 5 events samen getoond worden. Dus enkel de laatste 5 nieuwe events worden weergegeven op het scherm. Zodra er een nieuw event plaatsvindt, wordt het oudste event uit het scherm verwijderd indien deze limiet bereikt is. De gebruiker wordt op de hoogte gesteld van zowel de nieuwe als de oude events. Soms is het overzichtelijker om het scherm op te delen in meerdere schermen met groepen events op basis van een bepaalde eigenschap van de events met 'PARTITION BY' (Purich, 2009c). In onze case zou het bijvoorbeeld nuttig zijn om de events van elk gezin in een apart scherm te bekijken.

Voorbeeld: SELECT * FROM TravelBehaviour RETAIN 5 events PARTION BY HousholdID;

SELECT * FROM Travelbehaviour RETAIN 4 seconds;
Dit is een voorbeeld van een time-based sliding window. Elk event blijft 4 seconden op het scherm. Zodra de vier seconden om zijn, verdwijnt het event van het scherm. Het scherm toont dus enkel de nieuwe events die de laatste vier seconden hebben plaatsgevonden. Zowel de nieuwe als de oude events worden weer gerapporteerd aan de gebruiker. Het tijdsinterval kan uitgedrukt worden in dagen, uren, minuten, seconden of milliseconden afhankelijk van de toepassing. Soms moet dit tijdsinterval niet gebaseerd zijn op de werkelijke tijd, maar op een

timestamp, één van de eigenschappen van de events (Purich, 2009c). Dit wordt weergegeven in het volgende voorbeeld.
Voorbeeld: SELECT * FROM TravelBehaviour RETAIN 4 seconds BASED ON TravelTime;

SELECT * FROM TravelBehaviour RETAIN BATCH OF 4 seconds;
Ten slotte zien we hier een voorbeeld van een time-based batched window. De nieuwe events komen het scherm binnen wanneer er nieuwe verplaatsingen of events plaatsvinden. De verplaatsingen die binnen de batch van t + 4 seconden arriveren, worden samen getoond aan de gebruiker als nieuwe verplaatsingen wanneer deze t + 4 seconden verstreken zijn. Zodra de volgende vier seconden, t + 8, verstreken zijn, worden deze verplaatsingen geregistreerd als oude events aan de gebruiker. De query wordt dus enkel toegepast wanneer het scherm vol is in tegenstelling tot de andere sliding windows. Deze batch kan ook toegepast worden op een row-based window (Purich, 2009c).

Soms wil een gebruiker enkel speciale events zien verschijnen op het scherm zoals events met de grootste waarden 'LARGEST', events met de kleinste waarden 'SMALLEST of events met unieke waarden 'UNIQUE'. Hiervoor kan men 'WITH' gebruiken en specifiëren welke events in het scherm moeten verschijnen (Purich, 2009c). Zo kan bijvoorbeeld gezocht worden naar de personen die er het langst over doet om op het werk te geraken in de ochtend.

Voorbeeld: SELECT HouseholdID, PersonID, AVG(LeaveHTime-ArriveWSTime) AS GemiddeldeReisTijd FROM Travelbehaviour RETAIN 5 events WITH LARGEST GemiddeldeReisTijd;

5.6 Filters

Een inkomende datastroom bevat heel wat overbodige of onnuttige events die geen betekenis hebben. Deze events moeten uit de datastroom gefilterd worden. Er zijn twee methoden om data te filteren waarbij de plaats van de filter varieert. (Punich, 2009c)

SELECT * FROM (SELECT * FROM TravelBehaviour WHERE Activity = 'S') RETAIN 5 EVENTS;
Dit is een voorbeeld van de eerste methode, het gebruik van een subquery. Op deze manier worden de events gefilterd voordat ze op het scherm verschijnen. Deze filterwijze krijgt de voorkeur omdat zo de hoeveelheid data waarop de rest van de query uitgevoerd moet worden, geminimaliseerd wordt (Purich, 2009c).

SELECT * FROM TravelBehaviour RETAIN 5 EVENTS WHERE Activity = 'S';

De tweede methode maakt gebruik van het ‘WHERE’-commando in de query. Elk event verschijnt wel eerst in het scherm, maar de gebruiker wordt enkel op de hoogte gesteld van de events die voldoen aan de voorwaarde in het ‘WHERE’-commando. Op deze manier wordt de query wel op elk event uitgevoerd, wat dus meer tijd en werk vraagt (Purich, 2009c).

HOOFDSTUK 6. StreamSQL

6.1 Algemeen

StreamSQL is de tweede continue querytaal die aan bod komt. Deze querytaal werd ontwikkeld door StreamBase, een organisatie die zich bezig houdt met complex event processing. Deze organisatie ontwerpt software voor systemen die real-time datastromen analyseren en die hierop gepast reageren om onmiddellijk beslissingen te kunnen nemen (StreamBase, n.d.). StreamSQL is een onderdeel van het StreamBase CEP-platform, ontwikkeld om te kunnen werken met een enorme hoeveelheid real-time applicaties.

Vroeger werden vooral C++ en Java gebruikt als standaard tools voor real-time applicaties. Het probleem met deze tools is echter dat deze een lange ontwikkelingscyclus hebben. StreamSQL is een betere oplossing: gebruiksvriendelijk, flexibel en gemakkelijk uitbreidbaar voor ontwikkelaars. StreamSQL is zo ontworpen dat de queries zowel op datastromen als op databases uitgevoerd kunnen worden (StreamBase, n.d.).

Een eerste reden hiervoor is dat StreamSQL gebaseerd is op SQL. SQL is een gekende, gemakkelijke querytaal die gebruik maakt van relationele operators. De combinatie van de functionaliteit, de kracht en de gebruiksvriendelijkheid van SQL zorgen er voor dat deze querytaal de ideale basis vormt voor StreamSQL. StreamSQL heeft SQL uitgebreid zodat deze toegepast kan worden op continue datastromen (StreamBase, n.d.).

StreamSQL heeft de data manipulation language van SQL uitgebreid zodat het mogelijk wordt om een opgeslagen tabel te updaten met behulp van events uit een datastroom. Op deze manier wordt het mogelijk om elk deel van een datastroom op te slaan, zodat deze verder gebruikt kunnen worden. Het commando ‘INSERT INTO’ wordt hiervoor gebruikt. Dit wordt in 6.4 bij de commando’s verder uitgelegd (StreamBase, n.d.).

Daarenboven heeft StreamSQL allerlei opties die uitgevoerd kunnen worden op datastromen. Het filteren, het samenvoegen en combineren van datastromen zijn hier enkele voorbeelden van. Het scherm bepaalt op hoeveel gegevens de queries worden uitgevoerd met behulp van sliding windows (zie 2.6 sliding windows) en wanneer een operatie moet afgebroken worden om de output weer te geven (StreamBase, n.d.).

Met behulp van StreamSQL kunnen causale verbanden en patronen ontdekt worden. Op deze manier kunnen er relaties tussen events aan het licht komen die gebaseerd zijn op een tijdelijke

ordening. Ten slotte kan er gemakkelijk nieuwe procesfunctionaliteit in het systeem ontwikkeld worden aangezien de operators van StreamSQL gemakkelijk uitgebreid kunnen worden. Andere gespecialiseerde mogelijkheden kunnen eenvoudig toegevoegd worden vermits er al C++ en Java code is ingebouwd in StreamBase (StreamBase, n.d.).

6.2 StreamBase Studio

De StreamBase Studio is de tool van StreamBase waarmee gebruikers een real-time applicatie kunnen ontwikkelen met behulp van een grafische eventflow language of StreamSQL. De studio biedt de gebruiker een interface met heel veel mogelijkheden om een real-time applicatie te ontwikkelen, te monitoren, te testen en te integreren (StreamBase Systems, 2010).

De basis van de StreamBase studio is de StreamBase server. Deze server werkt niet zoals een traditionele database, waarbij gegevens eerst worden opgeslagen alvorens ze verwerkt worden door er queries op uit te voeren. StreamBase verwerkt deze gegevens, meer bepaald de events, echter real-time waardoor resultaten op het moment dat ze geproduceerd worden, verschijnen. Elk event kan meerdere keren voorkomen en wordt telkens opgeslagen als een uniek gegeven, een 'tuple'. Dit komt overeen met een rij in een traditionele database (StreamBase Systems, 2010).

Het 'SB Authoring perspective' van StreamBase is te vinden in bijlage 5. Een perspectief is een verzameling van verschillende schermen om het beëindigen van een specifieke taak te vergemakkelijken. De verschillende schermen zijn de 'package explorer', 'document editor', 'palette view', 'properties view', 'outline view' en ten slotte het 'typecheck errors view'. De 'package explorer' geeft de verschillende projecten weer die gebruikt worden door de StreamBase studio. De gebruiker kan hierdoor gemakkelijk navigeren tussen de projecten (StreamBase Systems, 2010).

De StreamBase applicaties kunnen vervolgens op drie verschillende manieren ontworpen worden in de 'document editor' van de studio. Eerst en vooral kan men deze applicaties grafisch ontwerpen met behulp van een grafische eventflow language door componenten aan elkaar te linken in de 'EventFlow Editor'. Daarnaast kan men de applicaties met behulp van de StreamSQL language ontwerpen in het scherm 'StreamSQL Editor'. StreamSQL is de tekstgebaseerde taal om CEP-applicaties te creëren. Als derde optie kan men een combinatie van beide technieken gebruiken. Een eventflow applicatie kan omgezet worden in StreamSQL. Andersom werkt dit converteerproces niet. Wanneer iemand voor de eerste keer werkt met StreamBase is het aan te raden om eerst te vertrekken van de eventflow taal om StreamSQL te begrijpen vermits de

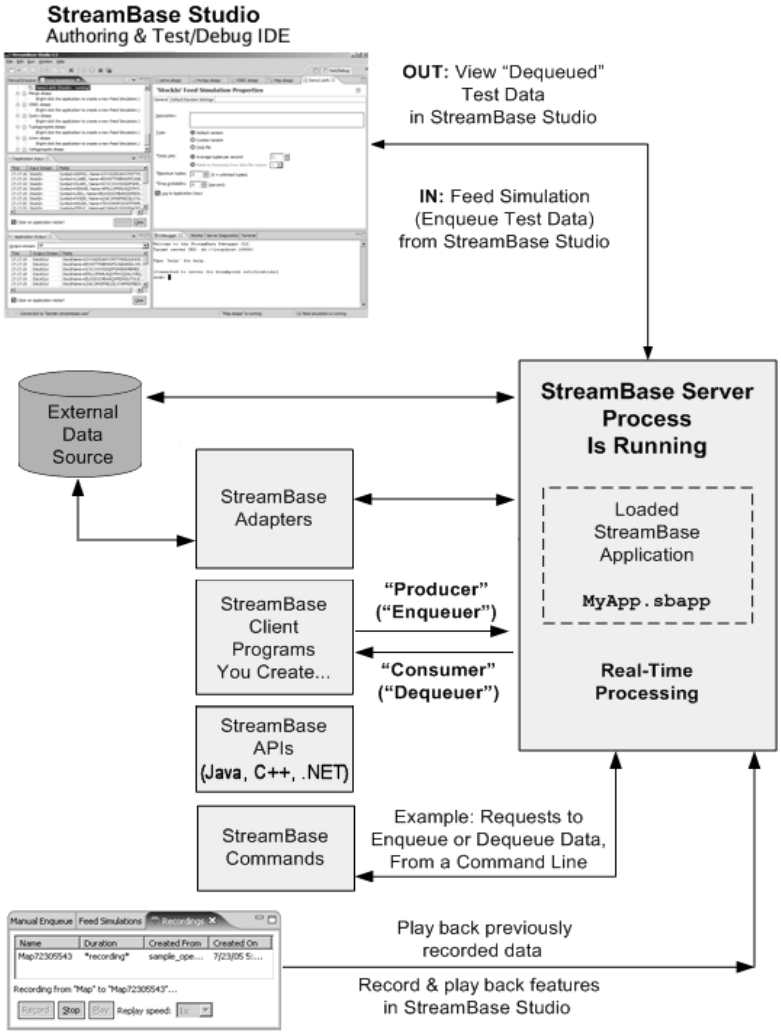
eventflow duidelijker weergeeft wat er exact gebeurt terwijl StreamSQL een abstractere codetaal is (StreamBase Systems, 2010).

De 'palette view' zorgt er daarnaast voor dat de gebruiker gemakkelijk componenten naar de 'event flow editor' kan slepen. In de 'properties view' kunnen de eigenschappen van de component, die op dat moment geselecteerd is, eenvoudig bepaald en aangepast worden. De 'outline view' geeft dan weer een overzicht van alle componenten, stromen, operators, ... in alfabetische volgorde zodat de gebruiker snel en efficiënt de gewenste component kan selecteren. De 'typecheck error view' informeert de gebruiker ten slotte over fouten in een openstaande eventflow of streamSQL project (StreamBase Systems, 2010).

Daarenboven is het belangrijk dat de data op een correcte manier getransformeerd wordt zodat er gemakkelijk mee gewerkt kan worden in de applicatie. Elke event wordt gekarakteriseerd door een aantal velden zoals het gezin (HouseholdID), het gezinslid (PersonID), de soort verplaatsing (TravelActivity) en het tijdstip van de verplaatsing (TravelTime). De events moeten op een goede manier georganiseerd worden, een goede benaming krijgen en elk veld moet in het juiste type (bijvoorbeeld Integer, String of Timestamp) worden geregistreerd. Het veld 'PersonID' krijgt bijvoorbeeld het type 'String' omdat het hier over tekst gaat, terwijl het veld 'TravelTime' het type 'Timestamp' krijgt aangezien het hier over een tijdsaanduiding gaat. De namen van alle velden, in combinatie met hun datatype, wordt het schema van een event genoemd.

Waar komen deze gegevens nu vandaan? Er moeten één of meerdere inputstromen gedefinieerd worden waarvan de gegevens naar de StreamBase server stromen. Er zijn verschillende mogelijke bronnen van gegevens. Op dezelfde manier moet een outputstroom ook opgevangen worden door een bepaald 'sink'. Figuur 9 toont de verschillende manieren waarop een datastroom een StreamBase applicatie kan binnenkomen en hoe een datastroom de StreamBase applicatie kan verlaten (StreamBase Systems, 2010).

Deze stappen, het aanmaken van een inputstroom, een inputbron definiëren en de data gepast transformeren, moeten allemaal doorlopen worden voordat de eigenlijke commando's kunnen uitgevoerd worden op de data, bijvoorbeeld het filteren van data. Zoals eerder vermeld kunnen er zowel eventflow applicaties als StreamSQL applicaties ontworpen worden. Deze masterproef beperkt zich tot StreamSQL applicaties. Deze StreamSQL applicaties worden opgeslagen als een tekstfile met een ssql-extensie. StreamSQL zal voor vele programmeurs bekend voorkomen aangezien deze querytaal veel weg heeft van SQL. Wat StreamSQL anders maakt is de mogelijkheid om commando's uit te voeren op een stroom van continu veranderende data (StreamBase Systems, 2010).



Figuur 9: Data in en uit een streambase applicatie (StreamBase Systems, 2010)

Een project kan in StreamBase enkel gerund worden indien de gebruiker zich in het 'test- en debug perspective' bevindt. In dit perspectief blijft de 'document editor' centraal staan, maar de andere schermen worden vervangen door een 'manual input/feed simulation view' en een 'application input/output view'. Deze schermen verschijnen niet allemaal naast elkaar, maar worden in tabbladen weergegeven (StreamBase Systems, 2010).

Met behulp van het 'manual input/feed simulation view' kan de gebruiker ofwel zelf events handmatig ingeven in een inputstroom ofwel de link leggen met een adapter die data aan een inputstroom levert via een 'feed simulation'. In de application 'input/output view' worden de events in de input- en outputstromen weergegeven die vooraf gedefinieerd waren (StreamBase Systems, 2010).

Ten slotte zijn er nog twee belangrijke functies: de 'StreamBase monitor' en de 'StreamBase manager'. Deze zorgen er voor dat de gebruiker het verloop van de uitvoering van een project kan analyseren aan de hand van statistieken en grafieken. De verschillende schermen worden uitvergroot weergegeven in bijlage 6.

6.3 Conventies, gereserveerde woorden en datatypes

De conventies van StreamSQL komen grotendeels overeen met deze van EPL (StreamBase Systems, 2010) :

- Sleutelwoorden worden steeds in HOOFDLETTERS weergegeven. Dit zijn gereserveerde woorden (zie bijlage 3) in StreamSQL, die niet voor een ander doeleinde gebruikt mogen worden.
- Vervangbare items worden in *kleine, schuin gedrukte letters* weergegeven.
- [Optionele items worden weergegeven tussen haakjes]
- Een keuze wordt uitgedrukt met een verticale streep | tussen 2 items die tussen (haakjes) staan
- Indien er meerdere optionele inhoud mogelijk is, wordt dit aangeduid door [,...]
- Commentaar wordt voorafgegaan door -- Dit zijn dus geen queries, maar deze commentaar kan wel meer duidelijkheid scheppen over de queries.
- Na elke query volgt een ; als afsluiting van die query;
- Wildcards: * wordt gebruikt om alle velden samen te selecteren.
- Wanneer een (dynamische) variabele, een veld, gedefinieerd wordt, moet ook het type van de variabele vastgelegd worden. Tabel 3 geeft alle mogelijke types in StreamBase weer.

Tabel 3: Data in en uit een streambase applicatie (StreamBase Systems, 2010)

Blob	Boolean	Double	Int	List
Long	String	Timestamp	Tuple	

6.4 Commando's

Er zijn heel wat commando's die uitgevoerd kunnen worden met behulp van StreamSQL. De DML commando's komen grotendeels overeen met EPL. Hier volgt een uiteenzetting van de belangrijkste commando's. Het commando wordt telkens vermeld met een korte uitleg bij. Vervolgens wordt er per commando een voorbeeld gegeven op de dataset 'het verplaatsingsgedrag van gezinsleden' (zie hoofdstuk 4). Deze voorbeelden komen ook vaak

terug in de casestudie in hoofdstuk 8, waarin de verschillende queries aan elkaar gekoppeld gaan worden. Deze casestudie schept heel wat duidelijkheid over de interpretatie van de queries.

Eerst en vooral worden de belangrijkste DDL-commando's uitgelegd, die helpen om de data te definiëren. Het creëren van input- en outputstromen, het schema van events, tabellen en het definiëren van een window behoren tot deze categorie.

CREATE INPUT STREAM *stream_identifier named_schema_identifier*|*anonymous_schema;*
Opdat de server queries kan uitvoeren op events, moet er eerst een inputstroom gecreëerd worden die de server voedt met een datastroom. In deze stap wordt de inputstroom gedefinieerd met behulp van de *'stream_identifier'* door de inputstroom een geschikte naam te geven. Vervolgens wordt het schema van de inputstroom bepaald. Als het schema vooraf gedefinieerd is (zie volgende alinea), wordt gewoon de naam van het schema achter de naam van de inputstroom geplaatst, de *'named_schema_identifier'*. Indien het schema nog niet gedefinieerd is, moeten de velden gespecificeerd worden samen met het datatype tussen haakjes achter de naam van de inputstroom, het *'anonymous_schema'*. Op deze manier wordt het schema gedefinieerd tegelijk met de creatie van de inputstroom (StreamBase Systems, 2010).

Het volgende voorbeeld maakt een inputstroom aan in de case waarbij het schema nog niet vooraf bepaald is. Een aantal belangrijke opmerkingen die telkens weer gaan terug komen: de gereserveerde woorden staan in hoofdletters, de velden met hun datatype staan tussen haakjes gescheiden met komma's en de query wordt afgesloten met ;. Telkens wanneer er nu verwezen wordt naar deze inputstroom wordt de naam 'TravelBehaviour' gebruikt. Het tweede voorbeeld geeft de datastroom weer waarbij er niets veranderd is aan de gegevens van 'LeaveHTime', 'ArriveWSTime', 'LeaveWSTime' en 'ArriveHTime'. Beide stromen worden verder gebruikt in de casestudies.

Voorbeeld:

```
CREATE INPUT STREAM TravelBehaviour(
HouseholdID          LONG,   PersonID      INT,  Activity      STRING,
PersonAge            INT,    Sex           INT,  HouseholdID   INT,
TravelActivity       STRING, TravelTime    INT,  Mode          INT,
ChildTransportation  INT,    Cars          INT,  Morning       STRING,
WorkStatus           INT);

CREATE INPUT STREAM TravelBehaviour2(
HouseholdID          LONG,   PersonID          INT,  Activity      STRING,
PersonAge            INT,    Sex               INT,  LeaveHTime    INT,
ArriveWSTime         INT,    LeaveWSTime       INT,  ArriveHTime   INT,
```

```
Mode                INT,    ChildTransportation  INT,  Cars        INT,
Morning             STRING, WorkStatus           INT);
```

CREATE SCHEMA *schema_identifier named_schema_identifier*|*anonymous_ schema*;

Om een inputstroom te creëren met een vooraf gedefinieerd schema, moet dit schema in een afzonderlijke query bepaald worden. Met behulp van het commando 'CREATE SCHEMA' wordt er een schema aangemaakt. Ook hier kan weer verwezen worden naar een vooraf gedefinieerd schema of kunnen de velden met hun datatype zelf gespecificeerd worden tussen haakjes. Dit commando wordt eerst en vooral gebruikt om een beter overzicht te bewaren over de queries door niet te veel informatie in één query weer te geven, maar door de informatie te spreiden over meerdere queries. Daarnaast is het ook handig om een schema op voorhand te bepalen indien het voor meerdere stromen, windows of tabellen gebruikt wordt. Op deze manier kan telkens gewoon naar de naam van het schema verwezen worden (StreamBase Systems, 2010). Het volgende voorbeeld creëert een inputstroom met een vooraf gedefinieerd schema. Eerst wordt het schema 'Travel' aangemaakt, om daarna een inputstroom 'TravelBehaviour' te creëren die verwijst naar dit schema 'Travel'. Let weer op de conventies.

Voorbeeld:

```
CREATE SCHEMA Travel (
HouseholdID         LONG,   PersonID            INT,  Activity      STRING,
Age                 INT,    Sex                 INT,  HouseholdID   INT,
TravelActivity      STRING, TravelTime          INT,  Mode          INT,
ChildTransportation INT,    Cars                INT,  Morning       STRING,  WorkStatus  INT
);
CREATE INPUT STREAM TravelBehaviour Travel;
```

CREATE OUTPUT STREAM *stream_identifier* WITH OUTPUT PREDICATE *predicate* AS *stream_ expression;*

Dit is een voorbeeld van hoe een outputstroom gecreëerd kan worden met StreamSQL. Een outputstroom kan op verschillende manieren gecreëerd worden, maar er zijn een aantal essentiële elementen die zeker in het commando moeten opgenomen worden. De '*stream_identifier'* geeft een naam aan de outputstroom. De '*stream_expression'* is een query die resulteert in een stroom en de '*predicate'* bevat de voorwaarde(n) waaraan de events moeten voldoen om te worden opgenomen in de outputstroom. Meestal worden deze commando's over meerdere queries verspreid. De outputstroom kan eerst aangemaakt worden met behulp van 'CREATE OUTPUT STREAM'. Achteraf kunnen de gegevens van een bepaalde query dan opgenomen worden in deze outputstroom met behulp van het 'INTO'-commando. Dit commando wordt later verder uitgelegd bij de DML-commando's (StreamBase Systems, 2010). Het volgende voorbeeld creëert twee outputstromen: één voor het verplaatsingsgedrag van gezinnen met één ouder en één voor het verplaatsingsgedrag van gezinnen met twee ouders. Het schema van deze stroom moet niet gespecificeerd worden vermits er automatisch wordt

aangenomen dat het schema van de outputstroom overeenkomt met het schema van de inputstroom. Indien dit niet het geval zou zijn, moet het schema uiteraard wel vermeld worden. Zoals eerder uitgelegd worden meestal de commando's over meerdere queries verspreid, hier wordt enkel de outputstroom gedefinieerd, de andere commando's komen later aan bod.

Voorbeeld: CREATE OUTPUT STREAM OneParent;
CREATE OUTPUT STREAM TwoParents;

CREATE STREAM *stream_identifier* `named_schema_identifier`|(`anonymous_schema`);

De input- en de outputstroom zorgen er voor dat events de applicatie binnenkomen en verlaten, maar soms is het ook nodig om een tussenstroom te voorzien in de applicatie zelf: 'CREATE STREAM'. Deze stromen kunnen niet gebruikt worden door externe explicaties, enkel voor de CEP-applicatie zelf. De stroom krijgt ook weer een naam en een schema toegewezen (StreamBase Systems, 2010). Stel dat een gebruiker nu de stromen van éénoudergezinnen, 'OneParent', en tweeoudergezinnen, 'TwoParents', nog verder wil filteren op bijvoorbeeld het transportmiddel. Dan moeten deze niet als een outputstroom gedefinieerd worden maar als een gewone stroom binnen het systeem zoals bij het volgende voorbeeld. Zo kunnen er nieuwe queries uitgevoerd worden op deze stroom. Het schema blijft hetzelfde als bij de inputstroom.

Voorbeeld: CREATE STREAM OneParent Travel;
CREATE STREAM TwoParents Travel;

CREATE [MEMORY|DISK] TABLE *table_identifier* `named_schema_identifier`|(`anonymous_schema`)
PRIMARY KEY (*field_identifier [,...]*);

Er kan daarnaast ook een tabel aangemaakt worden, analoog aan een relationele database, om gegevens tijdelijk of permanent op te slaan met behulp van 'CREATE TABLE'. De naam van de tabel wordt voorgesteld door de *'table_identifier'*. Indien events tijdelijk worden opgeslagen, dit wil zeggen totdat de StreamBase server afgezet wordt, wordt er een 'MEMORY TABLE' aangemaakt. Indien de events in een 'DISK TABLE' worden opgeslagen, kan de tabel verder gebruikt worden in een volgende sessie nadat de server is afgesloten. Bij de DML-commando's wordt verder gebruik gemaakt van deze aangemaakte tabel. Het schema van de tabel kan tussen haakjes bepaald worden, *'anonymous_schema'*, of dit kan net zoals bij een stroom een vooraf gedefinieerd schema, *'named_schema_identfier'*, zijn. De *'field_identifier'* geeft aan welk veld, welke combinatie van velden of welk schema gebruikt wordt als 'PRIMARY KEY', het primaire sleutelattribuut van de tabel. Elke tabel moet een primair sleutelattribuut hebben zodat elke rij als uniek beschouwd kan worden (StreamBase Systems, 2010). Veronderstel dat we een tabel willen creëren van personen die de auto gebruiken als transportmiddel en dat deze gegevens enkel tijdelijk worden opgeslagen. De tabel krijgt de naam 'Car' en het schema

'Travel' van de inputstroom wordt gewoon overgenomen. Het schema 'Travel' wordt gebruikt als primaire sleutel.

Voorbeeld:
```
CREATE MEMORY TABLE Car Travel
    PRIMARY KEY (Travel);
```

CREATE WINDOW *window_identifier* (*window_specification*);

Naast het creëren van de input- en outputstroom en het schema van de events is het af en toe ook zinvol om een window te definiëren. Met behulp van een window, kunnen queries uitgevoerd worden op een deel van heel de datastroom (zie 2.6 sliding windows). Zoals bij een stroom moet het window eerst een unieke naam krijgen, de '*window_identifier'*. In de '*window_specification'* worden dan de kenmerken van het window gedefinieerd zoals de grootte van het scherm. De grootte van het scherm is ofwel gebaseerd op een vast tijdsinterval, een maximum aantal events (zie 2.6 sliding windows) ofwel een specifieke waarde van een veld van de inputstroom. Dit moet ook allemaal vastgelegd worden in de query. Deze kenmerken worden in de volgende alinea uitgelegd. Op deze manier wordt een oneindige, inkomende stroom in verschillende segmenten opgedeeld. Het window kan ook op ongeveer dezelfde manier gedefinieerd worden achter het DML-commando 'FROM'. Het apart definiëren van het window krijgt wel de voorkeur, vermits het dan in meerdere queries gebruikt kan worden door de naam in een query te verwerken (StreamBase Systems, 2010).

SIZE *size* ADVANCE *increment*
{TIME | TUPLES | ON *field_identifier_w*}
[PARTITION BY *field_identifier_p* [,…]]
[OFFSET *offset*]

Dit is een standaardvorm met bijkomende opties van een '*window_specification'*. Met 'SIZE' wordt de grootte van het window bepaald op basis van het aantal events, een tijdsinterval of een specifieke waarde van een bepaald input veld. Met behulp van 'ADVANCE' wordt de hoeveelheid waarmee het scherm vooruit gaat, bepaald. Dit is meestal kleiner of gelijk aan de grootte van het window. 'TIME', 'TUPLES' of 'ON' wordt gekozen afhankelijk van of de grootte van het scherm gebaseerd is op tijd, een aantal events of een veldwaarde, '*field_identifier_w'*. Er kunnen ook meerdere schermen voor groepen events gecreëerd worden met 'PARTITION BY' op basis van een bepaald veld '*field_identifier_p'*. 'PARTITION BY' wordt enkel gebruikt als de grootte van een window bepaald wordt op basis van een aantal events. Het veld dat gespecificeerd wordt bij 'PARTITON BY' moet later ook bepaald worden bij 'GROUP BY' (zie SELECT-commando). Met 'OFFSET' kan bepaald worden wanneer het eerste window geopend wordt (StreamBase Systems, 2010). Het volgende voorbeeld creëert een scherm 'WindowTravelBehaviour' waar 5 events in komen. Per transportwijze wordt een nieuw scherm geopend. De naam 'WindowTravelBehaviour' kan nu in een query gebruikt worden zodat dit scherm op een bepaalde stroom van de query gezet wordt.

Voorbeeld: CREATE WINDOW WindowTravelBehaviour(
SIZE 50 ADVANCE 50 TUPLES
PARTITON BY Mode);

DECLARE *variable_identifier type* DEFAULT *default_value*
UPDATE FROM *stream query;*

Dit commando wordt gebruikt om dynamische variabelen te definiëren. De '*stream query'* kan een input-, output- of een gewone stroom zijn. Met behulp van de '*variable_identifier'* wordt er een naam gegeven aan de variabele en '*type'* geeft een datatype aan deze variabele (zie 6.3 conventies). Tijdens de initialisatie van de query kan er een beginwaarde aan de variabele toegekend worden met behulp van 'DEFAULT'. De waarde van deze variabele verandert afhankelijk van de '*stream query'*, dus tijdens de uitvoering van de query. Meestal wordt dit commando enkel gebruikt indien er meerdere inputstromen zijn. De dynamische variabele komt dan voort uit één stroom en wordt toegevoegd als een veld van een andere stroom (StreamBase Systems, 2010). Zo kan bijvoorbeeld een variabele aangemaakt worden die als een filter dient voor de leeftijd. De gebruiker moet dan in de query geen getallen gebruiken, maar kan deze zelf bepalen welke leeftijd als filter gebruikt wordt via een aparte stroom. De beginwaarde wordt op negen jaar gezet met behulp van 'DEFAULT'. Dit voorbeeld komt aan bod in het eerste project van hoofdstuk 8.

Voorbeeld: CREATE INPUT STREAM LeeftijdIngeven(NieuweLeeftijd INT);
DECLARE LeeftijdFilter INT DEFAULT 9
UPDATE FROM (SELECT LeeftijdIngeven.NieuweLeeftijd FROM LeeftijdIngeven);

Naast de DDL-commando's zijn er de DML- commando's die de events van de datastroom gaan manipuleren.

SELECT *target_list_entry*
FROM *event_source* | FROM PATTERN *template* WITHIN (*interval* TIME | *value* ON *field*)
[WHERE *predicate*]
[HAVING *predicate*]
[GROUP BY *field_identifier_grouping*]
[ORDER BY *field_identifier_ordering* [DESC] [LIMIT *number*]]
[INTO *stream_identifier*];

Het bekendste commando dat voorkomt is het 'SELECT'-commando. Dit commando wordt, net als bij SQL, ook veel gebruikt bij StreamSQL om één of meerdere velden te selecteren van events die aan een bepaalde voorwaarde voldoen. Om events uit een inputstroom te selecteren en deze weer te geven in een outputstroom, wordt er altijd een 'SELECT'-commando gebruikt. Zoals te zien in de algemene query zijn er heel wat uitbreidingen mogelijk van het 'SELECT'-commando. Er zijn nog heel wat meer mogelijkheden, maar deze zijn de belangrijkste. De '*target_list_entry'* geeft weer welke velden de query in het resultaat wil weergeven. Deze lijst

kan uit één of meerdere velden bestaan. Indien de gebruiker alle velden van de events uit de inputstroom in de outputstroom wil, wordt de wildcard * gebruikt. Deze velden moeten uit een bepaalde stroom, window of tabel geselecteerd worden, de '*event source*' met behulp van het 'FROM'-commando. Het alternatief, 'FROM PATTERN', komt in de volgende alinea aan bod. Het 'SELECT'-commando kan gebruikt worden om functies te berekenen over de stroom of om ongewenste events weg te filteren met behulp van het 'WHERE'-commando. Indien een event of een stroom niet aan de voorwaarde in het 'WHERE'-commando voldoet, wordt deze geëlimineerd. Wanneer deze query wordt uitgevoerd, ontstaat er een nieuwe stroom met de resultaten van de query in, de outputstroom. Met behulp van 'INTO' wordt aangegeven in welke outputstroom de resulterende events van de query terechtkomen. Dit is heel belangrijk aangezien de resulterende events anders niet in het outputscherm verschijnen (StreamBase Systems, 2010). Bij de DDL-commando's zijn er al voorbeelden gegeven om een inputstroom TravelBehaviour en twee stromen OneParent en TwoParents te creëren. Met behulp van 'SELECT...FROM...WHERE…' kan er nu heel eenvoudig een link gelegd worden tussen de inputstroom en de twee stromen. Op deze manier worden er events van de inputstroom naar de outputstromen gestuurd. De events worden nu ingedeeld afhankelijk van of het over een éénoudergezin of een tweeoudergezin gaat.

```
SELECT * FROM TravelBehaviour WHERE HouseholdType = 1 INTO OneParent;
SELECT * FROM TravelBehaviour WHERE HouseholdType = 2 INTO TwoParents;
```

Indien een gebruiker daarnaast naar een patroon wilt zoeken in een stroom kan er, in plaats van het 'FROM'-commando, het 'FROM PATTERN'-commando gebruikt worden. In de '*template*' wordt de volgorde van het patroon waarnaar de query gaat zoeken, bepaald. Dit wordt in project 3 en project 4 van de casestudie in hoofdstuk 8 wel duidelijk hoe dit juist gebeurt. Met behulp van het 'WITHIN'-commando wordt het tijdsinterval bepaald waarin gezocht wordt naar dit patroon. Dit kan op basis van het huidige tijdstip met behulp van 'TIME' of dit kan op basis van een timestamp in één van de velden van de events met behulp van 'ON'. Het volgende voorbeeld komt uit project 3 (hoofdstuk 8) en wordt daar verder uitgelegd.

```
Voorbeeld: SELECT Man.HouseholdID, Man.ChildTransportation AS ChildTransportationMan,
           Vrouw.ChildTransportation AS ChildTransportationVrouw
           FROM PATTERN (Man THEN Vrouw)
           WITHIN 86400 ON TravelTime
           WHERE  Man.HouseholdID = Vrouw.HouseholdID AND (Man.ChildTransportation = 1
                  OR Man.ChildTransportation = 2) AND Vrouw.ChildTransportation = 4
           INTO Out;
```

Er zijn daarenboven een aantal zogenaamde aggregatie functies die op gegevens uitgevoerd kunnen worden. Dit zijn een soort verzamelfuncties die kort samengevat gegevens in één

waarde weergeven. Een gemiddelde (AVG) berekenen, een maximum (MAX) of een minimum (MIN) en het aantal rijen gegevens tellen (COUNT) zijn hier voorbeelden van. In bijlage 4 zijn de belangrijkste aggregatie functies opgesomd. Deze formules kunnen niet gecombineerd worden met een 'WHERE'-commando. Een 'HAVING'-commando biedt een oplossing hiervoor. 'HAVING' zorgt er voor dat er aan een query voorwaarden in verband met aggregatie functies toegevoegd kunnen worden. Dit commando werkt als een soort beperking op welke events geselecteerd worden. Het commando kan echter enkel gebruikt worden indien er een window is aangemaakt (StreamBase Systems, 2010).

Een ander belangrijk commando dat samen gaat met het gebruik van een aggregatie functie in het 'SELECT'-commando, en dus met een window, is het 'GROUP BY'-commando. De resulterende events worden met behulp van dit commando gegroepeerd op basis van een veldnaam. Elke groep events die hierdoor ontstaat, krijgt een eigen scherm gebaseerd op de velden gespecifieerd in het 'GROUP BY'-commando en het 'PARTITION BY'-commando. Het is dan ook heel belangrijk dat het veld van 'GROUP BY' en 'PARTITION BY' overeenkomt (StreamBase Systems, 2010). Het volgende voorbeeld selecteert het 'HouseholdID', de gemiddelde reistijd in de ochtend en de wijze van transport van het window 'WindowTravelBehaviour' dat hiervoor gedefinieerd is in het voorbeeld. De events die een gemiddelde reistijd hebben van minder dan 20 minuten worden geselecteerd. De verwijzing naar het window moet tussen vierkante haken worden geplaatst achter de naam van de stroom. Dit voorbeeld komt terug in casestudie 5 (hoofdstuk 8) waarin de toepassing van windows aan bod komt.

Voorbeeld: SELECT AVG(ArriveWSTime-LeaveHTime) AS OchtendTijd, Mode FROM TravelBehaviour2 [WindowTravelBehaviour] GROUP BY Mode HAVING OchtendTijd < 20;

Het is soms niet voldoende om gewoon gegevens terug te krijgen als resultaat. Het gebeurt dat een gebruiker een bepaalde ordening wilt in de resulterende events. Met behulp van 'ORDER BY' kunnen de resulterende gegevens van een bepaalde query gerangschikt worden op basis van een gespecificeerd veld van de datastroom. Resultaten kunnen zowel in oplopende 'ASC' als aflopende 'DESC' volgorde geordend worden. Wanneer de manier van ordenen niet expliciet bepaald is, wordt er vanuit gegaan dat de resultaten in oplopende volgorde geordend worden. Indien de resultaten in aflopende volgorde moeten verschijnen, wordt er 'DESC' achter geplaatst. Een mogelijke uitbreiding van 'ORDER BY' is 'LIMIT' waarbij bepaald wordt hoeveel events er per waarde van een bepaald veld gehouden worden, '*number*' (StreamBase Systems, 2010). Hier volgt een eenvoudig voorbeeld waarbij de stroom geordend wordt op basis van het 'HouseholdID':

Voorbeeld: SELECT * FROM TravelBehaviour
ORDER BY HouseholdID;

CREATE *stream_identifier*
INSERT INTO *table_identifier '(*'*column_identifier* [, ...]')'
SELECT *target_list_1* FROM *stream_expression*
[RETURNING *target_list_2*
INTO *stream_identifier*]*;*

Indien er nieuwe events toegevoegd moeten worden aan een bestaande tabel, wordt het 'INSERT INTO'-commando gebruikt voor het 'SELECT'-commando. De velden die gespecifieerd worden in *'target_list_1'* van een bepaalde datastroom *'stream_expression'* worden toegevoegd aan de tabel met de naam *'table_identifier'*. De *'column_identifier'* geeft aan in welke velden de nieuwe data worden geplaatst. De velden die vermeld worden in *'target_list_2'* worden aan de datastroom *'stream_identifier'* toegevoegd. Op deze manier worden de veranderingen aan de tabel ook doorgegeven aan een datastroom en wordt de gebruiker dus op de hoogte gesteld van deze veranderingen. Indien het 'RETURNING'-commando niet aanwezig is in deze context, gaan deze veranderingen onopgemerkt aan de tabel voorbij (StreamBase Systems, 2010). De tabel 'Car' die eerder is aangemaakt kan met dit commando dus gegevens ontvangen. Er wordt ook een outputstroom 'UpdateTabel' aangemaakt waarin de events die aan de tabel worden toegevoegd terecht komen.

Voorbeeld: CREATE OUTPUT STROOM UpdateTabel Travel;
INSERT INTO Car
SELECT * FROM TravelBehaviour
WHERE (Mode = 4 OR Mode = 5)
RETURNING *
INTO UpdateTabel;

REPLACE INTO *table_identifier* (*column_identifier*[, ...]) *stream_expression*
[RETURNING *target_list* INTO *stream_identifier*];

In plaats van een event te updaten in een tabel, kan een event ook verwijderd en vervangen worden door een nieuw event met behulp van het 'REPLACE INTO'-commando. Events van de tabel *'tabel_identifier'* worden vervangen indien het event aan de *'stream_expression'* voldoet. De veranderingen kunnen weer in een datastroom weergegeven worden met behulp van 'RETURNING' (StreamBase Systems, 2010). Stel bijvoorbeeld dat er een voorwaarde, waaraan de events moeten voldoen, wordt toegevoegd alvorens de events in de tabel opgeslagen worden. Zo kan bijvoorbeeld 'WorkStatus' = 2 worden toegevoegd. Hierdoor blijven enkel de events over van personen die voltijds werken en die met de auto naar het werk gaan.

Voorbeeld: REPLACE INTO Car
SELECT * FROM TravelBehaviour

```
WHERE (Mode = 4 OR Mode = 5) AND WorkStatus = 2
RETURN * INTO UpdateTabel;
```

BSORT {*stream_identifier_1* | *stream_expression*} ON *field_identifier*
SIZE *size*
INTO *stream_identifier;*

Het kan voorkomen dat een gebruiker de events niet op chronologische volgorde nodig heeft, maar in een andere ordening. De volgorde van events kan gewijzigd worden met behulp van 'BSORT'. De '*stream_identifier'* of '*stream_expression'* (uitdrukking die resulteert in een stroom) bepaaldt welke stroom herschikt wordt. Met behulp van 'ON' wordt bepaald op basis van welk veld '*field_identifier'* de stroom opnieuw gerangschikt wordt. 'SIZE' geeft de hoeveelheid events weer die gesorteerd moeten worden en met 'INTO' wordt gespecifieerd in welke stroom de geordende events terecht komen. De inputstroom 'TravelBehaviour' kan bijvoorbeeld gesorteerd worden op het veld 'HouseholdID'. Momenteel staan de events van deze stroom in chronologische volgorde. Dit gebeurt in het volgende voorbeeld voor de eerstvolgende 200 events en deze geordende events komen terecht in de stroom 'TravelBehaviourSorted'.

Voorbeeld:
```
CREATE STREAM TravelBehaviourSorted Travel;
    BSORT TravelBehaviour ON HouseholdID
    SIZE 200
    INTO TravelBehaviourSorted;
```

APPLY JAVA "*java_class_name*" '('*parameter_list*')'
[AS *adapter-name*]
FROM *stream_identifier* | INTO *stream_identifier*;

Om een adapter toe te voegen aan een input- of een outputstroom wordt het 'APPLY'-commando gebruikt. Met behulp van dit commando kan een JAVA-klasse waarin de zogenaamde 'embedded adapters' aangemaakt zijn, opgevraagd worden. Elk soort adapter heeft een andere naam, *'java_class_name'*, en vereist andere parameters, *'parameter_list'*. Indien de haakjes rond de parameters tussen aanhalingstekens staan, wil dit zeggen dat de haakjes verplicht zijn. Aan deze adapter kan een korte, gemakkelijke naam worden toegewezen met behulp van 'AS *adapter-name'*. Indien het gaat om een inputadapter wordt de inputstroom achter het 'INTO'-commando bepaald. Als het om een outputadapter gaat, wordt de outputstroom achter het 'FROM'-commando gespecifieerd. De twee JAVA-klassen die voor de casestudie belangrijk zijn, zijn deze van een CSV input en output adapter, namelijk:

- "com.streambase.sb.adapter.csvreader.CSVReader" voor de inputadapter
- "com.streambase.sb.adapter.csvwriter.CSVWriter" voor de outputadapter

(StreamBase Systems, 2010)

Een goed alternatief voor het gebruik van een inputadapter is de 'feed simulation' die eerder is uitgelegd. Dit gaat eerder gebruikt worden in de casestudie, vermits het gebruik van het 'APPLY JAVA'-commando redelijk omslachtig is zoals blijkt uit het volgende voorbeeld. Het voorbeeld maakt een outputadapter aan die terug zal komen in het eerste project van hoofdstuk 8. De parameters zijn overgenomen van een voorbeeld in het programma.

Voorbeeld van een outputadapter:

```
APPLY JAVA "com.streambase.sb.adapter.csvwriter.CSVWriter" AS OutputAdapterTravel
(IncludeHeaderInFile = "true", MaxRollSecs = "0", StringQuoteOption = "Quote if necessary",
FlushInterval = "1", StringQuoteCharacter = "\"", RollPeriod = "None", FieldDelimiter = ",",
TsFormat = "false", CompressData = "false", MaxFileSize = "0",OpenOutputFileDuringInit =
"false", ThrottleErrorMessages = "false", NullValueRepresentation = "null", FileName =
"outputstroom.csv", IfFileDoesntExist = "Create new file", CheckForRollAtStartup = "false",
SyncOnFlush = "false", IfFileExists = "Append to existing file")
FROM KindFiltered;
```

GATHER [*stream_identifier_1*.]*field_identifier_1* AS *output_field_identifier_1*[, ...]
FROM *stream_identifier_1*, *stream_identifier_2*[, ...]
USING *field_identifier_gather*
INTO *stream_identifier;*

In plaats van één inputstroom, kan het ook voorkomen dat er meerdere inputstromen zijn die gecombineerd kunnen worden tot één outputstroom. Hier zijn meerdere commando's mogelijk zoals het 'GATHER'-commando. 'GATHER' combineert inputstromen op basis van events van deze stromen die een bepaald veld gemeenschappelijk hebben. De outputstroom wordt dan gecreëerd met de waarden van de inkomende events. De *'field_identifier_1'* geeft aan welk veld of welke velden weergegeven moeten worden in de outputstroom. Deze velden kunnen van elke inkomende stroom afkomstig zijn. Indien een veld gespecifieerd wordt dat afkomstig kan zijn van meerdere inputstromen, kan de gewenste inputstroom aangegeven worden voor het veld met behulp van *'stream_identifier_1.'.* Op deze manier kan er geen verwarring ontstaan tussen de verschillende stromen. Achter het 'FROM'-commando worden alle inputstromen (*stream_identifier_1, stream_identifier_2,…*) gedefinieerd waaruit de events voortkomen. Achter het commando 'USING' wordt vervolgens bepaald op basis van welk veld, '*field_identifier_gather'* de inkomende events gefilterd worden. Het is belangrijk dat dit veld in elke inputstroom voorkomt. Zodra in elke inputstroom de events hebben plaatsgevonden met een gemeenschappelijke waarde voor dat veld, wordt een output event gecreëerd met de vooraf gedefinieerde velden (StreamBase Systems, 2010). Het kan bijvoorbeeld dus voorkomen dat de data van het verplaatsingsgedrag niet afkomstig zijn van een stroom, maar van drie stromen: een voor de kinderen, de mannen en de vrouwen.

Voorbeeld: GATHER Kind.HouseholdID, Kind.Mode, Man.ChildTransportation,
Vrouw.ChildTransportation FROM Kind, Man, Vrouw
USING HouseholdID INTO Stream1;

MERGE *stream_expression_1*, *stream_expression_2*
USING *field_identifier_m* [GROUP BY *field_identifier_g*]
INTO *stream_identifier*;

Een tweede optie bij het gebruik van meerdere inputstromen is 'MERGE' waarbij alle events van twee inkomende stromen, *'stream_expression_1'* en *'stream_expression_2'* samengevoegd worden tot een outputstroom. Bij dit commando is het wel essentieel dat de schema's van beide inputstromen overeenkomen. De events in de outputstroom worden geordend op basis van een gedefinieerd veld *'field_identifier_m'* met 'USING'. Het is vaak ook zinvol om de events eerst te groeperen, 'GROUP BY', op basis van een bepaald veld *'field_identifier_g'*. Deze groepering gebeurt voordat de twee inputstromen samengevoegd en geordend worden (StreamBase Systems, 2010). Stel dat er twee inputstromen zijn, zowel voor de kinderen als voor de ouders. Dan kunnen deze gegevens samengevoegd worden in 1 stroom, geordend op basis van het 'HouseholdID'.

Voorbeeld: MERGE Kind, Ouders
USING HouseholdID
INTO Stream1;

stream_expression_1 UNION *stream_expression_2* [UNION ...]
INTO *stream_identifier;*

Als laatste kan 'UNION' gebruikt worden om meerdere inputstromen samen te voegen. Op deze manier worden ook alle events van de inkomende stromen in één outputstroom weergegeven. Het verschil met 'MERGE' is dat hier de outputstroom niet geordend wordt (StreamBase Systems, 2010). Hier kan opnieuw het voorbeeld van de drie stromen (Kind, Man, Vrouw) aangehaald worden.

Kind UNION Man UNION Vrouw INTO Stream1;

Aan elk DML-commando kan ten slotte nog een extra commando worden toegevoegd achteraan de query: 'ERROR INTO *stream_identifier'.* Dit commando laat toe dat elke error die voorkomt, behandeld wordt als data. De fouten worden dan opgeslagen in de gedefinieerde stroom '*stream_identifier'*. Deze fouten kunnen zowel veroorzaakt worden door de data zelf (Bv. in een bewerking delen door nul) als door externe resources (Bv. het openen van een onbeschikbaar bestand). Dit commando kan erg belangrijk zijn om specifieke fouten te detecteren en allemaal in eenzelfde stroom weer te geven om deze te analyseren en op te lossen (StreamBase Systems, 2010).

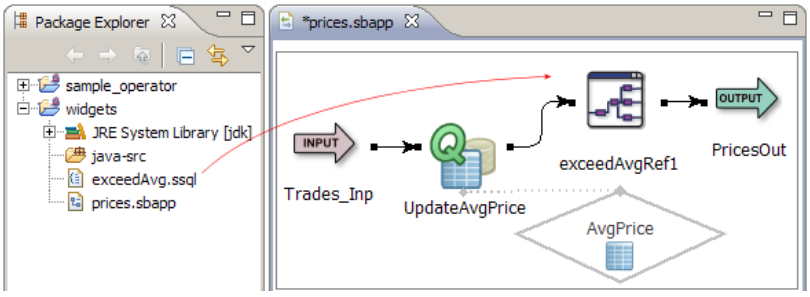
6.5 Functies

Naast de commando's zijn er ook nog heel wat belangrijke functies die in de queries kunnen voorkomen. Er zijn twee soorten functies: simpele functies en aggregatie functies. Simpele functies zijn functies die telkens op één event uitgevoerd worden. Aggregatie functies worden net als bij SQL op een hele reeks data uitgevoerd om één waarde terug te krijgen (StreamBase Systems, 2010). In bijlage 7 is een lijst te vinden van alle mogelijke aggregatie functies en simpele functies. Indien er een functie gebruikt wordt in de casestudie, wordt deze daarin uitgelegd.

6.6 Voordelen van StreamBase Studio

Waarom is de StreamBase Studio een goede tool om continue queries toe te passen op een datastroom met behulp van StreamSQL?

- De software omvat een StreamSQL editor waardoor het gemakkelijker wordt om StreamSQL queries te schrijven. Het is heel gemakkelijk om sleutelwoorden in te voegen door middel van de 'Content Assist' die de mogelijke sleutelwoorden weergeeft. Vervolgens controleert StreamBase continu de syntax van de queries, die de gebruiker ingeeft, op fouten.
- De software is zo ontworpen zodat StreamSQL applicaties gemakkelijk gecombineerd kunnen worden met andere StreamBase componenten zoals een Event Flow applicatie. Op deze manier kan de gebruiker zijn/haar voorkeur van aanpak gebruiken.
- Doordat een Event Flow en StreamSQL applicatie gecombineerd kunnen worden, kan er heel wat tijd bespaard worden in zowel de ontwikkeling als het onderhoud van de applicatie. Zoals te zien in het voorbeeld in figuur 10 kan een StreamSQL applicatie gewoon in het scherm van de EventFlow Editor gesleept worden (StreamBase Systems, 2010).



Figuur 10: Voorbeeld van een combinatie van StreamSQL en EventFlow applicatie (StreamBase Systems, 2010

HOOFDSTUK 7. Continuous Computation Language

7.1 Algemeen

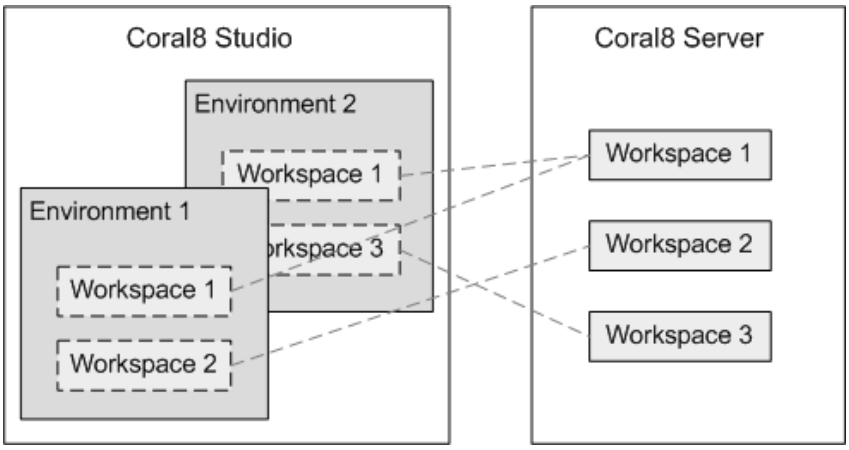
Continuous Computation Language of kortweg CCL is de continue querytaal die gebruikt wordt door Coral8 om datastromen real-time te verwerken en te analyseren. Deze querytaal is, net zoals StreamSQL en EPL, gebaseerd op SQL zodat het normaalgezien voor een doorsnee SQL-gebruiker eenvoudig is om deze taal aan te leren staat op de website van zowel StreamBase als Coral8 te lezen (www.streambase.com & www.aleri.com). CCL heeft SQL enkel uitgebreid zodat continue datastromen verwerkt kunnen worden onder andere door opties toe te voegen voor windowing, output controle en het herkennen van patronen. De redenen waarom StreamSQL een betere oplossing is t.o.v. Java en C++ zijn ook van toepassing op CCL. CCL is gebruiksvriendelijk, flexibel en gemakkelijk uitbreidbaar voor ontwikkelaars.

7.2 Coral8 studio en server

De belangrijkste elementen van het CEP-platform van Coral8 zijn de Coral8 studio en de Coral8 server. De Coral8 studio is het grafische deel van het Coral8-platform waarin Coral8-projecten ontwikkeld, gecompileerd en uitgevoerd worden. De code die ontwikkeld wordt in de Coral8 studio, wordt vervolgens gecompileerd. De resultaten hiervan worden doorgestuurd naar de Coral8 server die zorgt voor de uitvoering van de code. Identiek aan de StreamBase studio kunnen er zowel grafisch als met een querytaal projecten ontwikkeld worden. De querytaal die gebruikt wordt door Coral8 is de continuous computation language, CCL (Aleri, 2009).

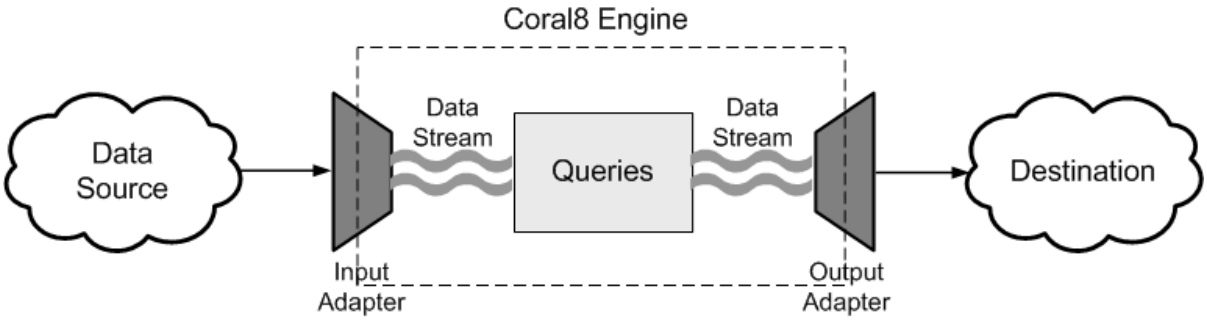
De Coral8 studio bestaat uit een aantal omgevingen die verbonden zijn met één of meerdere werkruimtes van de Coral8 server. Werkruimtes worden gebruikt om projecten te organiseren, terwijl een omgeving enkel wordt aangemaakt in de Coral8 studio zodat een gebruiker verschillende delen van de werkruimtes kan zien en bewerken. Figuur 11 geeft deze relatie tussen de studio en de server weer. In deze figuur is duidelijk te zien dat de studio en de server gescheiden applicaties zijn waartussen verbinding gemaakt moet worden door middel van een werkruimte. Op deze manier is er interactie tussen de studio en de server mogelijk. Beide applicaties, de studio en de server, moeten dus apart opgestart worden voordat er projecten ontwikkeld kunnen worden. Een werkruimte is dan ook niet vast bepaald voor een bepaald project. Eenzelfde project kan, telkens als het geopend wordt, gekoppeld worden aan een andere werkruimte van de Coral8 server. Het is ook belangrijk om een project expliciet af te

sluiten, want het afsluiten van de Coral8 studio zorgt er niet voor dat het project stopt op de server (Aleri, 2009).



Figuur 11: Relatie tussen de Coral8 studio en de server (Aleri, 2009)

Een Coral8 project bestaat vervolgens uit een aantal componenten typisch voor een CEP-applicatie, zoals te zien op figuur 12. Data uit één of meerdere inputbronnen stroomt binnen via een inputadapter. Deze inputadapter zorgt er voor dat de data in het juiste formaat getransformeerd wordt, geschikt voor het Coral8-platform. Op deze manier vormen de events een inputstroom die verwerkt kan worden door er queries op uit te voeren. De resultaten van deze queries vormen de events voor de outputstroom, die door de outputadapter terug geconverteerd worden naar een formaat, geschikt voor de eindbestemming van de data. Een project wordt opgeslagen met extensie .ccl (Aleri, 2009).



Figuur 12: Componenten van een project (Aleri, 2009)

Het platform is ontworpen zodat er continue queries uitgevoerd kunnen worden op een inkomende datastroom. De query wordt eenmalig doorgegeven aan de server en deze blijft de

query uitvoeren op de inkomende data totdat de server effectief gestopt wordt door de gebruiker (Aleri, 2009).

Bijlage 8 is een voorbeeld van de 'user interface' van de Coral8 studio. Deze bestaat, net zoals bij StreamBase, uit verschillende schermen: 'explorer view', 'editor view', 'properties view', 'flow view' en 'output view'. In de 'explorer view' worden de verschillende omgevingen en projecten weergegeven. Van elk project worden de verschillende objecten, zoals de datastromen en adapters, weergegeven. Dit scherm wordt gebruikt om deze projecten of objecten te selecteren en er acties op uit te voeren. De 'editor view' is het werkblad waarin de queries worden ingegeven en aangepast (Aleri, 2009).

Daarnaast geeft de 'properties view' alle eigenschappen van het object, dat op dat moment geselecteerd is, weer. Indien de gebruiker niet gewoon is om met queries te werken, kan deze de eigenschappen van bijvoorbeeld een inputstroom handmatig ingeven. Zo worden deze eigenschappen automatisch omgezet in een query. De 'flow view' geeft het stroomdiagram weer van de componenten van het project, een grafische representatie van het project. Dit is handig voor de gebruiker doordat de queries visueel worden voorgesteld. Hier kunnen ook componenten gecreëerd worden. Zowel tijdens de ontwikkeling als tijdens de uitvoering is dit scherm erg nuttig. Dit scherm helpt om de project componenten te visualiseren en hoe deze in interactie staan met elkaar. Het helpt daarnaast ook om logische errors te lokaliseren. Wanneer de gebruiker de muis op een component plaatst, verschijnt de querycode. Bij elke component staat informatie over het project zoals 'latency', 'memory usage' en 'message traffic'. Ten slotte levert de 'output view' belangrijke boodschappen aan de gebruiker over de compilatie en de uitvoering van het project, bijvoorbeeld fouten in de queries (Aleri, 2009).

Het grote voordeel van de Coral8 studio is dat de verschillende schermen met elkaar verbonden zijn. Wanneer de gebruiker een object selecteert in de 'explorer view', worden de overeenkomstige zaken in alle andere schermen geselecteerd. Indien de gebruiker daarnaast zaken aanpast in bijvoorbeeld de 'editor view', worden de andere schermen automatisch geüpdatet (Aleri, 2009).

Wanneer een project gerund wordt, verschijnt er een nieuw scherm in plaats van de 'properties view': de 'status view'. Dit scherm geeft heel wat informatie weer over het project zoals de status van het project ('runned' of 'stopped'), het tijdstip waarop het project gestart wordt, de totale hoeveelheid geheugen dat gebruikt wordt en informatie over de events. Daarnaast worden er nieuwe schermen geopend bij de uitvoering van een project: 'stream of window viewer' om de stromen en windows te analyseren. Zo worden de events getoond die arriveren in een bepaalde stroom en wordt de huidige inhoud van een window weergegeven. Deze worden

wel gescheiden van heel het ontwikkelingsscherm geopend. De verschillende schermen worden vergroot weergegeven in bijlage 9 (Aleri, 2009).

7.3 Conventies, datatypes en operatoren

De conventies en de datatypes komen weer ongeveer overeen met EPL (Aleri, 2009) :

- GERESERVEERDE WOORDEN worden in hoofdletters weergegeven om deze te benadrukken. Deze woorden zijn terug te vinden in bijlage 10 en mogen niet voor een ander doeleinde gebruikt worden.
- Vervangbare items worden in *kleine, schuin gedrukte letters* weergegeven.
- [Optionele items worden weergegeven tussen haakjes]
- Een keuze wordt uitgedrukt met een verticale streep |tussen 2 items die tussen haakjes staan
- Indien er meerdere optionele inhoud mogelijk is, wordt dit aangeduid door [,...]
- Na elke query volgt een ; als afsluiting van die query;
- /*commentaar*/ of --Commentaar kan de gebruiker aanwenden om queries te verduidelijken.
- Wildcards: * wordt gebruikt om alle velden samen te selecteren.
- De datatypes komen grotendeels overeen met deze van StreamSQL zoals te zien in tabel 4.

Blob	Boolean	Float	Integer	Interval
Long	String	Timestamp	Tuple	XML

Tabel 4: Datatypes CCL (Aleri, 2009)

- De belangrijkste operatoren die gebruikt kunnen worden in een CCL-query zijn weergegeven in tabel 5. Meer uitleg over deze operatoren is te vinden in bijlage 2.

Operator	**Betekenis?**
+,-	Plus, min
^	Macht
*,/, mod	Vermenigvuldiging, deling, rest
=, !=, <>, <, >, <=, >=	Vergelijkingen
LIKE, REGEXP_LIKE, IN, IS NULL, IS NOT NULL	Soortgelijk
AND, OR, XOR, NOT	Logische operatoren

Tabel 5: CCL operatoren (Aleri, 2009)

7.4 Commando's

Bij het bestuderen van StreamSQL en EPL valt onmiddellijk op dat deze commando's heel wat gelijkenissen vertonen. Dit is ook zo bij CCL. Vandaar dat de CCL commando's enkel kort aan bod komen en er vooral de nadruk gelegd wordt op de commando's die verschillen met deze van StreamSQL en EPL.

De opbouw van de queries bestaat weer uit drie grote delen, namelijk: het creëren van een inputstroom, het uitvoeren van queries op deze inputstroom en de resultaten weergeven in een outputstroom.

CREATE [INPUT|OUTPUT| LOCAL] STREAM *name [SCHEMA clause];*
Met behulp van dit commando worden de eerste en de laatste stap, de input- en de outputstroom, aangemaakt. 'CREATE INPUT STREAM' en 'CREATE OUTPUT STREAM' zijn net dezelfde commando's als bij StreamSQL. Een interne stroom, die enkel in de applicatie zelf gebruikt wordt, creëren, gebeurt met het commando 'CREATE LOCAL STREAM'. Het verschil met streamSQL ligt in het definiëren van het schema in de 'SCHEMA *clause'*. Achter de naam, '*name'*, van de stroom kan het schema bepaald worden met 'SCHEMA'. In de '*clause'* worden dan de velden van de stroom samen met hun datatype gespecifieerd (Aleri, 2009).

Voorbeeld: CREATE INPUT STREAM TravelBehaviour
SCHEMA(

HouseholdID	LONG,	PersonID	INTEGER,	Activity	STRING,
PersonAge	INTEGER,	Sex	INTEGER,	HouseholdType	INTEGER,
TravelActivity	STRING,	Traveltime	INTEGER,	Mode	INTEGER,
ChildTransportation	INTEGER,	Cars	INTEGER,	Morning	STRING,
Workstatus	INTEGER)	;			

CREATE SCHEMA *name {(col_name type)* |INHERATES [FROM] *schema_name(col_name)};*
Natuurlijk kan het schema ook afzonderlijk aangemaakt worden om niet te veel informatie in één query weer te geven of om het schema meermaals te gebruiken. Het eerste deel van de query komt ongeveer overeen met StreamSQL waarbij de naam, *'name'*, van het schema gedefinieerd wordt met tussen haakjes de velden van elk event, *'col_name',* met hun datatype '*type'*. Het is belangrijk dat elk schema een unieke naam krijgt. Optioneel kan het 'INHERATES FROM'-commando gebruikt worden om te verwijzen naar een vooraf gedefinieerd schema '*schema_name'* binnen de module. De velden van dit schema worden dan als de eerste velden beschouwd van het nieuw schema. Indien de gebruiker nog andere velden wil toevoegen, kunnen deze gespecifieerd worden tussen haakjes '*col_name'* (Aleri, 2009).

```
Voorbeeld: CREATE SCHEMA Travel(
             HouseholdID          LONG,     PersonID     INTEGER,  Activity        STRING,
             PersonAge            INTEGER,  Sex          INTEGER,  HouseholdType   INTEGER,
             TravelActivity       STRING,   Traveltime   INTEGER,  Mode           INTEGER,
             ChildTransportation  INTEGER,  Cars         INTEGER,  Morning        STRING,
             Workstatus           INTEGER)  ;
           CREATE INPUT STREAM TravelBehaviour
           SCHEMA Travel;
```

IMPORT *file;*

In de vorige query kan er enkel gebruik gemaakt worden van een vooraf bepaald schema indien dit schema effectief gedefinieerd is binnen een module. Wanneer een gebruiker toch een extern schema wil aanwenden, kan het 'IMPORT'-commando gebruikt worden. Op deze manier worden alle schema's en functies van een bestand, *'file'*, geïmporteerd in de huidige query module (Aleri, 2009).

Voorbeeld: IMPORT "../../mydir/MyModule1.ccl";

ATTACH{INPUT|OUTPUT}ADAPTER *name* TYPE *type* TO STREAM *stream* [PROPERTIES{*propertie_value*}];

'ATTACH ADAPTER' maakt een adapter aan zodat er data in de applicatie stromen. Met behulp van 'TO STREAM' wordt de stroom, *'stream'*, gedefinieerd. Aan deze stroom wordt de adapter verbonden. Het soort adapter wordt gedefinieerd door er 'INPUT' of 'OUTPUT' voor te plaatsen. Een inputadapter kan enkel verbonden worden met een inputstroom terwijl een outputadapter met eender welke stroom verbonden kan worden. Doordat er heel wat types adapters zijn, moet het 'TYPE' van de adapter gedefinieerd worden. Een voorbeeld van een type adapter is 'ReadFromCsvFileAdapterType' waarbij de gegevens in een speciaal Excel formaat zijn opgeslagen. Afhankelijk van het type adapter zijn er een aantal eigenschappen, 'PROPERTIES', die bepaald moeten worden. Een aantal eigenschappen die toegevoegd kunnen worden, zijn bijvoorbeeld: het bestand waarin de data staat, het aantal keer dat de events naar de stroom verzonden moeten worden, het aantal events per seconden en specificaties over de timestamp. De volgorde waarin de eigenschappen worden geplaatst, is niet belangrijk. Meestal wordt er achter een outputstroom nog een adapter geplaatst zodat de data terug geconverteerd kan worden naar een geschikt formaat. Op deze manier kan een externe applicatie hier gebruik van maken (Aleri, 2009).

```
Voorbeeld: ATTACH INPUT ADAPTER Travel TYPE ReadFromCsvFileAdapterType
           TO STREAM TravelBehaviour
           PROPERTIES
               FILENAME = "$ProjectFolder\data\CSV data travelbehaviour.csv",
               RATE     = "2",
```

```
USECURRENTTIMESTAMP  = "true",
TIMESTAMPCOLUMN      = "false";
```

CREATE VARIABLE *data_type name [=value];*

Met behulp van het commando 'CREATE VARIABLE' kan een gebruiker een variabele definiëren die enkel gebruikt wordt binnen de huidige query module. Eerst wordt het datatype van de variabele bepaald *'data_type'* en de naam, *'name'*. Indien gewenst, kan de gebruiker ook een initiële waarde, *'value'*, toekennen aan deze variabele. Dit commando is heel nuttig indien de gebruiker een waarde van een variabele wil veranderen. Een stroom kan gekoppeld worden aan deze variabele met behulp van het 'SET VARIABLE'-commando (zie later). Zo kan de gebruiker eenvoudig een boodschap via deze stroom zenden zodat de variabele verandert (Aleri, 2009). In een uitbreiding van de eerste casestudie wordt een filter geplaatst op de leeftijd van de personen. Door hier een variabele te gebruiken in plaats van een waarde, kan deze leeftijd door de gebruiker altijd en gemakkelijk worden aangepast.

Voorbeeld: CREATE VARIABLE INTEGER LeeftijdFilter;

CREATE WINDOW *win_name* SCHEMA *clause* {KEEP *clause* [**,** KEEP *clause*]}INSERT REMOVED [ROWS] INTO *name*]

Het creëren van een scherm, 'CREATE WINDOW', is één van de grote pluspunten van continue querytalen. Zo kan een gebruiker bepalen welke events en hoeveel events in het scherm moeten verschijnen met behulp van sliding windows (zie 2.6). Op de events die in het scherm verschijnen, kunnen dan opnieuw queries uitgevoerd worden. Het scherm krijgt een naam, '*win_name'*, en het schema geeft, net zoals bij een stroom, weer welke velden van elk event in het scherm verschijnen. 'KEEP' wordt gebruikt om aan te geven hoeveel events er in het scherm zullen verschijnen. Dit wordt verder uigelegd bij het 'SELECT'-commando. De oude events die het scherm terug verlaten, kunnen naar een stroom of een ander scherm geleid worden met behulp van 'INSERT REMOVED ROWS INTO *name'* (Aleri, 2009).

```
Voorbeeld: CREATE WINDOW WindowTravelBehaviour
           SCHEMA TravelSchema
           KEEP 20 ROWS Per Mode;
```

ON *name* [AS *alias*] [WHEN *trigger*] DELETE FROM *window_name* [WHERE *clause*]

'DELETE FROM' gaat expliciet rijen verwijderen uit een window, '*window_name'*. Met 'ON' definieert de gebruiker de stroom of het window waarvan de inkomende events er voor zorgen dat er rijen geëlimineerd worden van een ander window. Deze databron kan eventueel een

andere naam krijgen met behulp van 'AS *alias'*. Met behulp van 'WHEN' kan er optioneel een limiet geplaatst worden op het verwijderen van events. Een gebruiker kan bijvoorbeeld ingeven dat er pas vanaf een bepaalde waarde rijen verwijderd worden. Het 'WHERE'-commando werkt als een filter door te bepalen welke rijen verwijderd moeten worden (Aleri, 2009).

Voorbeeld:
```
ON STREAM
    DELETE FROM NW1
    WHERE StreamA.ID=NW1.ID;
```

INSERT INTO *name* VALUES (*column_expression [,…]) [,…]* OUTPUT *clause;*

'INSERT VALUES' doet het tegenovergestelde van het 'DELETE'-commando en gaat dus waarden creëren. Op een bepaald moment worden deze nieuwe waarden dan toegevoegd aan een stroom of een window, *'name'*. De waarden die worden toegevoegd worden bij *'column_expression'* bepaald. Het is belangrijk dat het datatype van de *'column_expression'* overeen komt met het datatype van het veld waarin de waarden worden toegevoegd. Indien er meerdere velden per event zijn, moet voor elk veld een waarde binnen de haakjes gespecifieerd worden. Het is ook mogelijk om meerdere rijen tegelijk toe te voegen [,…]. De 'OUTPUT *clause'* specificeert wanneer de waarden worden toegevoegd. Hier zijn heel wat mogelijkheden voor, onder andere OUTPUT EVERY 30 SECONDS, OUTPUT AFTER 20 SECONDS, OUTPUT AT STARTUP (Aleri, 2009).

Voorbeeld:
```
INSERT INTO Sites
    VALUES ('New York'),('London')
    OUTPUT AT STARTUP;
```

ON *Name* [WHEN *trigger*] SET *name* = *value* [,…];

Met het 'SET VARIABLE'-commando kan een waarde van een of meerdere variabelen veranderd worden. Met 'ON *name'* wordt de stroom bepaald waarin de waarde van deze variabele veranderd wordt. De nieuwe waarde, *'value'*, van de variabele, *'name'*, wordt bepaald achter het 'SET'-commando. Optioneel kan het 'WHEN'-commando gebruikt worden om de gevallen, in welke de query gebruikt wordt, te limiteren (Aleri, 2009). In het volgende voorbeeld wordt de waarde van de aangemaakte variabele 'LeeftijdFilter' (zie eerder) op een leeftijd van 12 jaar gezet.

Voorbeeld:
```
ON TravelBehaviour
    SET LeeftijdFilter = 12;
```

INSERT *clause*
SELECT *expression* [AS *alias*] FROM *stream*|*window* [KEEP *clause*]
[Matching *clause*]
[ON *clause*]
[WHERE *condition*]
[GROUP BY *column*]
[HAVING *Boolean*]
[ORDER BY *column* [ASC|DESC]]
[LIMIT *count* [ROW[S]] [OFFSET *skip* [ROW[S]] [PER *column*]]
[OUTPUT *clause*]*;*

Zodra de belangrijkste componenten in de applicatie gecreëerd zijn, kan er een 'SELECT'-commando uitgevoerd worden op de datastroom om betekenisvolle events te verkrijgen. Hierboven zijn de voornaamste uitbreidingen van het 'SELECT'-commando weergegeven (Aleri, 2009). 'SELECT', 'FROM', 'WHERE', 'GROUP BY', 'HAVING' en 'ORDER BY' komen overeen met deze van StreamSQL en EPL, die reeds uitgelegd zijn. Voor de uitleg van deze commando's worden dan ook naar hoofdstuk 5 en hoofdstuk 6 verwezen.

```
Voorbeeld: INSERT INTO LeaveHTimeFiltered
           SELECT *, AVG(Traveltime) FROM FilteredTravelBehaviour
           WHERE TravelActivity = 'LeaveHTime'
           GROUP BY Mode;
```

Het 'INSERT'-commando kan op twee verschillende manieren gebruikt worden. De eerste manier is 'INSERT INTO *name*', identiek aan de EPL query en 'INSERT' van StreamSQL. Een tweede manier is het gebruik van 'INSERT WHEN *condition* THEN *name* [,…] [ELSE *name*]'. Met behulp van dit commando kan één stroom opgedeeld worden in meerdere stromen aan de hand van een bepaalde voorwaarde, *'condition'*. De query wordt uitgevoerd op elk event. Zodra een event aan de voorwaarde voldoet, wordt deze naar de overeenstemmende stroom, *'name'*, gezonden (Aleri, 2009).

```
Voorbeeld: INSERT
              WHEN ChildTransportation = 1 THEN Dropoff
              WHEN ChildTransportation = 2 THEN Pickup
              WHEN ChildTransportation = 3 THEN DropoffAndPickup
           SELECT HouseholdID, PersonID, PersonAge, Sex, HouseholdType, Mode,
                  ChildTransportation, WorkStatus
           FROM TravelBehaviour;
```

De 'KEEP *clause'* komt overeen met het 'RETAIN' commando van EPL. Dit commando kan optioneel gebruikt worden bij het FROM-commando en wordt ook gebruikt bij het creëren van een window zoals eerder vermeld. Hier komt dus ook weer het begrip sliding windows naar voren. De verschillende toepassingen van sliding windows bij het 'KEEP'-commando zijn te

vinden in figuur 15, telkens met een voorbeeld. Het gebruik van een van deze mogelijkheden wordt het 'window policy' genoemd (Aleri, 2009).

Row-based sliding window	KEEP 3 ROWS	Telkens drie rijen houden
Time-based sliding window	KEEP 10 SECONDS	Elk event wordt tien seconden weergegeven
Row-based batched window	KEEP EVERY 3 ROWS	Events verschijnen en verdwijnen in groepjes van drie
Time-based batched window	KEEP EVERY 10 SECONDS	Events verschijnen en verdwijnen per interval van tien seconden
Keep All	KEEL ALL	Alle events die in een window terecht komen, blijven behouden

Tabel 6: Toepassing sliding windows CCL (Aleri, 2009)

Het 'LIMIT'-commando komt ook voor bij StreamSQL, maar wordt bij CCL iets anders gebruikt. Bij StreamSQL kan dit commando enkel voorkomen in combinatie met 'ORDER BY', terwijl bij CCL 'LIMIT' ook zonder 'ORDER BY' kan gebruikt worden. *'Count'* geeft het maximaal aantal events die gepubliceerd kunnen worden van een query weer. Soms is het ook nuttig voor de gebruiker dat de eerste events niet opgenomen worden in de resultaten door 'OFFSET' te gebruiken. Het aantal events dat niet gebruikt worden, wordt bepaald door de *'skip'*. Als de gebruiker de resulterende events toch wil groeperen op basis van deze limiet kan 'PER' gebruikt worden. Zo kan bepaald worden op basis van welke kolom, *'column'*, de events gegroepeerd worden (Aleri, 2009).

Voorbeeld:
```
INSERT INTO TwoParents
    SELECT * FROM TravelBehavour
    WHERE HouseholdType = 2
    GROUP BY TravelMode
    LIMIT 2 ROWS PER TravelMode;
```

'OUTPUT' zorgt voor synchronisatie, een limiet of vertraging op de output van een query. Dit commando komt altijd als laatste commando bij een query en is verplicht bij een 'INSERT VALUES'-commando. Zodra alle events gefilterd en verwerkt zijn door de andere commando's, bepaalt dit commando hoe de output wordt weergegeven (Aleri, 2009). De verschillende mogelijkheden zijn opgesomd in tabel 2.

Tabel 7: Output opties (Aleri, 2009)

OUTPUT EVERY	Interval van events of tijd waarna het meest recente of geen event gepubliceerd wordt
OUTPUT AFTER	Vertraging van alle output tot het tijdsinterval voorbij is en publiceert dan alle output samen
OUTPUT FIRST WITHIN	Publiceert enkel het eerste event dat gearriveerd is in een bepaald tijdsinterval
OUTPUT AT	Output publiceren op een specifiek tijdstip

```
Voorbeeld: INSERT INTO LeaveHTimeFiltered
           SELECT *, AVG(Traveltime) FROM FilteredTravelBehaviour
           WHERE TravelActivity = 'LeaveHTime'
           OUTPUT EVERY 10 SECONDS;
```

Een ander belangrijk optioneel commando is 'MATCHING'. Met dit commando kan een patroon van events met specifieke temporele relaties gedetecteerd worden. Dit commando volgt op het 'FROM'-commando. De databronnen, die in het 'MATCHING'-commando gebruikt worden, moeten overeen komen met de databronnen bij 'FROM'. Zo kunnen er meerdere databronnen doorgelicht worden op zoek naar een specifiek patroon van events die arriveren of niet arriveren binnen een specifiek tijdsinterval. De databronnen moeten in de juiste volgorde, volgens het patroon, achter dit commando geplaatst worden. De gebruiker kan ook op zoek gaan naar zogenaamde non-events, events die niet hebben plaatsgevonden. Om deze non-events te vinden moet er voor de datastroom een uitroepteken geplaatst worden(Aleri, 2009).

Aan een 'MATCHING'-commando kan ook een 'ON'-commando toegevoegd worden. In dit deel van de query kan een 'join' voorwaarde voor een patroon gespecificeerd worden. Dit commando bestaat uit één of meerdere gelijkheden (*source_column* = *source_column*). Zo gaat een query enkel naar een patroon zoeken in de rijen waarbij de gedefinieerde kolommen van de databronnen overeen komen (Aleri, 2009). Het volgende voorbeeld wordt verder uitgelegd in casestudie 3 en 4.

```
Voorbeeld: INSERT INTO StreamAlert
           SELECT Man.HouseholdID, Man.ChildTransportation, Vrouw.ChildTransportation
           FROM Man, Vrouw
           MATCHING [20 SECONDS: Man, Vrouw]
           ON Man.HouseholdID = Vrouw.HouseholdID
           WHERE (Man.ChildTransportation = 1 OR Man.ChildTransportation = 2) AND
           Vrouw.ChildTransportation = 4;
```

7.5 Voordelen

- Coral8 zorgt er voor dat veranderingen automatisch worden opgeslagen nadat deze zijn doorgevoerd
- De combinatie van de grafische weergave en de queries zorgt er voor dat de queries gemakkelijker te begrijpen zijn
- Indien de gebruiker een verandering invoert in een bepaald scherm, worden de andere schermen automatisch geüpdatet
- Zodra de gebruiker een woord begint te typen, worden onmiddellijk alle mogelijkheden weergegeven om het woord aan te vullen. Autocompletion werkt niet enkel bij

gereserveerde woorden, maar ook bij stromen en kolomnamen gedefinieerd door de gebruiker

- Kleurencodes worden gebruikt om verschillende delen van de syntax weer te geven: Gereserveerde woorden in het blauw, namen van kolommen, namen van stromen en numerieke waarden in het zwart, strings tussen dubbele aanhalingstekens in het rood en strings tussen enkele aanhalingstekens in het roos. Indien de gebruiker commentaar wil plaatsen tussen de queries, wordt deze in het groen weergegeven. Op deze manier is het gemakkelijker om een overzicht te bewaren en duidelijk weer te geven wat alles betekend in een query. Dit is zeker handig wanneer er veel queries nodig zijn en er dus een ingewikkelde structuur ontstaat (Aleri, 2009).

HOOFSTUK 8. Vergelijking platformen en querytalen

Er zijn veel gelijkenissen tussen de platformen en de querytalen, maar ook veel (subtiele) verschillen die de ene tool al wat aangenamer maakt ten opzichte van de andere. Deze algemene gelijkenissen en verschillen komen hier uitgebreid aan bod. In 8.3 wordt dan de vergelijkende studie voort gezet met de casestudie op de dataset 'verplaatsingsgedrag van gezinsleden'.

8.1 Algemene werking platformen

Elke querytaal hoort bij een bepaald platform. Alvorens de querytalen vergeleken kunnen worden, is het daarom belangrijk om de algemene werking van deze platformen te analyseren.

Heel wat bedrijven bieden een trial-versie aan van hun software op het internet. Deze kan normaal gemakkelijk van hun website afgehaald worden, samen met een licentie die geldig is voor 30 of soms zelfs 60 dagen. Bij StreamBase en Coral8 gaat dit allemaal heel eenvoudig en efficiënt door simpelweg op 'download software' te klikken en een gebruikersaccount op de website aan te maken. Dit is eigenlijk de hoofdreden waarom deze twee platformen gekozen zijn. Een tweede reden waarom deze twee platformen gebruikt zijn in deze masterproef is de beschikbaarheid van een uitgebreide handleiding over de querytalen. Dit is uiteraard noodzakelijk om de querytalen onder de knie te krijgen. Voor EPL van Oracle werd er geen trial versie van een platform online gevonden. Enkel de platformen van StreamBase en Coral8 worden daarom uitgebreid met elkaar vergeleken.

Een eerste grote verschil zit in het opstarten van de platformen. Terwijl StreamBase automatisch verbinding maakt met de server wanneer de gebruiker de StreamBase studio opstart, moet dit bij Coral8 afzonderlijk gebeuren. Zowel de server als de Coral8 studio moeten opgestart worden. De applicatie van Coral8 blijft dus runnen, ook wanneer de studio afgesloten wordt. Dit kan zowel een voordeel als een nadeel zijn. Zo kan het handig zijn om de server te laten draaien terwijl de studio afgesloten wordt. Het kan ook zijn dat de gebruiker de server per ongeluk vergeet af te sluiten waardoor deze onnodig blijft draaien. De Streambase server draait enkel wanneer de studio wordt opgestart.

De indeling van het basisscherm van de studio's is vervolgens verschillend. Terwijl bij Coral8 de eventflow en de queries gelijktijdig worden weergegeven in de 'editor view' en de 'flow view', worden deze bij StreamBase beide weergegeven in de 'document editor' als twee aparte tabbladen. Bij Coral8 zijn de schermen dan ook veel meer verbonden met elkaar dan bij

StreamBase. Zodra er in een bepaald scherm van Coral8 iets wordt toegevoegd of aangepast, worden de andere schermen onmiddellijk geüpdatet. Bij StreamBase maakt men ofwel een eventflow-diagram ofwel een StreamSQL-project. Een eventflow-diagram kan echter wel geconverteerd worden naar StreamSQL, maar dan moet deze opdracht expliciet gegeven worden aan de StreamBase studio. Andersom is dit niet waar en kan men een StreamSQL-project niet omzetten in een eventflow-diagram. De gebruiker kan dus niet, zoals bij Coral8, de queries ingeven en deze visueel volgen op een eventflow diagram. In dit opzicht is het voor een gebruiker eenvoudiger en overzichtelijker om met Coral8 te werken. Tabel 8 geeft een vergelijking weer over de schermen die ongeveer dezelfde functie hebben bij de twee platformen.

Tabel 8: Vergelijking scherm StreamBase en Coral8

StreamBase	Coral8
Package explorer	Explorer
StreamSQL Editor	Editor view
Event flow editor	Flow view
Properties view	Properties view
Typecheck errors, problems, console	Output view
SB Manager, SB Monitor	Flow view, Status view
Applicaton input/output	Stream, window view

Bij de StreamBase studio is het dus mogelijk dat de gebruiker zelf een eventflow-diagram aanmaakt in plaats van te werken met queries. Voor de gebruiker kan het gemakkelijker zijn om op deze manier om te gaan met event logs. Nadien kan dit diagram dan geconverteerd worden in StreamSQL om te begrijpen wat deze eventflow nu eigenlijk doet. In Coral8 kan de gebruiker ook zelf een eventflow-diagram aanmaken, maar wordt de nadruk eerder gelegd op het werken met queries. De andere schermen in Coral8, de ‘editor view’ en de ‘properties view’, worden dan onmiddellijk geüpdatet, wat het natuurlijk wel eenvoudiger maakt voor de gebruiker. Bij StreamBase moet deze link handmatig gelegd worden, terwijl dit bij Coral8 allemaal automatisch gebeurt. Dit komt ook terug wanneer het project gesaved moet worden. Het saven moet expliciet gebeuren bij StreamBase, terwijl Coral8 bij elke verandering automatisch het project savet. Elk project dat open staat in StreamBase en op dat moment niet gesaved is, wordt gekenmerkt door een *. Zodra de gebruiker het project savet, verdwijnt dit teken. Bij Coral8 moet de gebruiker een aantal seconden wachten totdat elke verandering aangepast wordt om verder te kunnen werken met de bijgewerkte versie van het project.

Wanneer een applicatie uitgevoerd wordt, is het scherm dat de gebruiker te zien krijgt heel verschillend voor beide platformen. Bij Coral8 blijft het scherm ongeveer hetzelfde. De

gebruiker moet wel expliciet nieuwe schermen openen om de input- en outputstromen, 'view all streams in module', en eventuele windows, 'view named windows', weer te geven. Deze worden, indien gewenst, netjes naast elkaar getoond met de functie 'arrange all viewers'. Zo kan de werking tussen de verschillende stromen gemakkelijk geanalyseerd worden. Het nadeel is wel dat het scherm waarin alles ontwikkeld wordt en het scherm waarin alles uitgevoerd wordt eigenlijk apart verschijnen. Bij StreamBase echter moet een ander perspectief geopend worden, namelijk het 'test- en debugperspectief' zoals eerder uitgelegd. Dit opent automatisch door op de 'run' knop te duwen. De input- en outputstromen worden met tabbladen weergegeven en verschijnen dus niet naast elkaar wat de analyse van de verschillende stromen omslachtig maakt. De gebruiker kan eenvoudig navigeren tussen de verschillende stromen door op de tabbladen te klikken en de queries blijven naast de datastromen zichtbaar waardoor de link tussen een stroom en een query gemakkelijk geanalyseerd kan worden.

Daarnaast verschillen de resultaten van een uitgevoerd project tussen de twee platformen. Het kan natuurlijk interessant zijn voor een gebruiker om de statistieken van een project te kunnen volgen terwijl dit project uitgevoerd wordt. Bij Coral8 verschijnen deze statistieken naast de componenten in de 'flow view' en veranderen deze mee terwijl het project uitgevoerd worden. Er wordt bijvoorbeeld weergegeven hoeveel events er telkens in een component stromen en hoeveel events er uit een component stromen en dus door een filter geraakt zijn. Bij StreamBase worden enkel de datastromen weergegeven en hoeveel events er in een bepaalde stroom terecht komen. Om statistieken te analyseren kan als eerste optie de 'StreamBase Monitor' geopend worden in de zogenaamde 'Streambase Command Prompt', speciaal ontwikkeld voor Windows gebruikers. Door het commando 'sbmonitor' te geven, worden er statistieken over het lopende project weergegeven. Een tweede optie is het openen van de 'StreamBase Manager' in het platform zelf. Hierin wordt ongeveer hetzelfde getoond als in de 'flow view' en de 'status view' van Coral8. De statistieken van beide mogelijkheden gaan wel enkel over de input en de output. Wat er in de applicatie zelf gebeurt, wordt niet weergegeven bij StreamBase. Zodra een project afgebroken wordt, verdwijnen de statistieken van de 'SB monitor' bij StreamBase, terwijl bij de 'SB manager' en Coral8 de statistieken blijven staan. In bijlage 11 worden de statistieken van Coral8 met StreamBase vergeleken in een voorbeeld.

Eén van de meest gebruikte inputadapters bij beide studio's is daarenboven een CSV-bestand, een Comma Separated Values bestand. De gegevens van de casestudie zijn ook omgezet naar een CSV-bestand (zie hoofdstuk 4) zodat dit bestand als inputadapter gebruikt kan worden. Met behulp van dit bestand wordt een artificiële continue datastroom gecreëerd. Het is een speciaal bestand in Excel, waarbij alle kolommen in één kolom worden samengevoegd, gescheiden door komma's. Een voorbeeld van een CSV-bestand van de data is te vinden in bijlage 12. Ook voor de outputadapters wordt er telkens een CSV-bestand gebruikt. Deze vangen dan de events op die in de outputstroom terecht komen.

De events van een datastroom komen binnen op een bepaald tijdstip, de zogenaamde 'Timestamp' zoals eerder vermeld. Dit kan bij beide platformen op twee manieren: het huidige tijdstip kan als timestamp gebruikt worden of de inputadapter kan een kolom bevatten waarin de timestamp bepaald is. Indien het huidige tijdstip als timestamp gebruikt wordt, moet de gebruiker ook bepalen hoeveel events er per seconde in de applicatie binnenstromen. Maar als de inputadapter een timestamp kolom bevat, moet dit bij coral8 de eerste kolom zijn en moet het specifieke formaat --/--/---- 00:00:00 gebruikt worden. Bij StreamBase kan de gebruiker selecteren welke kolom gebruikt wordt als timestamp en het formaat van de timestamp kan zelf bepaald worden waardoor je ook enkel met een tijdstip of een datum kan werken. Zo is er meer kans dat de timestamp van een bepaalde adapter in het juiste formaat staat, terwijl dit bij Coral8 niet het geval is.

Indien een bepaald project snel getest moet worden om te kijken of het juist werkt, kan het hele proces versneld worden zodat er sneller output gevormd wordt. Bij een project in Coral8 is hier een speciale functie voor 'accelerate playback' in de 'properties view'. Hiermee kan de gebruiker bepalen hoe snel het project uitgevoerd wordt. Om een project te versnellen bij StreamBase, moet bij de eigenschappen van de feed simulation 'As fast as possible' aangevinkt worden bij de datasnelheid. Op deze manier kan een hele dataset verwerkt worden op enkele seconden. Dit is vooral handig bij historische data en om het project te testen alvorens het in productie gaat. Er moet wel opgepast worden dat dit niet met te veel data gedaan wordt, zodat de applicatie alles verwerkt krijgt.

Zowel StreamBase als Coral8 zorgen er ten slotte voor dat een nieuwe gebruiker gemakkelijk de eerste stappen kan zetten in het CEP-platform met behulp van goede documentatie die beschikbaar is op internet en in de help-functie van de studio. Aan de hand van tutorials geraak je onmiddellijk vertrouwd met het platform. Ook de querytalen worden uitgebreid uitgelegd met een voorbeeld om de theoretische uitleg te verduidelijken. De voorbeelden bij StreamBase zijn wel beperkter als bij Coral8 en soms ontbreken er cruciale voorbeelden om bepaalde theorie te begrijpen.

In onderstaande tabel 9 wordt een overzicht gegeven van de voorstaande vergelijking tussen de algemene werking van StreamBase en Coral9.

Tabel 9: Vergelijking algemene werking platformen

Wat?	StreamBase	Coral8
Trialversie beschikbaar?	Ja	Ja
Opstarten platform	Server start automatisch samen met de studio	Server en studio apart opstarten
Indeling basisscherm	Tabbladen	1 geheel

Project opslaan	Handmatig opslaan	Automatisch
Flow diagram vs. queries	Aparte projecten, flow diagram kan wel geconverteerd worden naar queries	1 geheel aan elkaar gelinkt
Uitvoering project	Test- en debugperspectief openen	Stream viewer openen
Statistieken	- SB Monitor of SB Manager - Enkel over input en output	- Flow view - Over alle stromen en windows
Gebruikte inputadapter	CSV-bestand	CSV-bestand
Gebruikte outputadapter	CSV-bestand	CSV-bestand
Timestamp	- Willekeurige kolom van het bestand - Verschillende formaten mogelijk	- Eerste kolom van het bestand - "----/--/-- uu:mm:ss"
Versneld testen	As fast as possible	Accelerate playback
Helpfunctie	Uitgebreid aanwezig	Uitgebreid aanwezig

8.2 Vergelijking queries

Na de algemene werking van de platformen is het tijd om het gebruik van de queries te vergelijken met elkaar. In tabel 10 worden de queries weergegeven van de drie querytalen die hetzelfde doel hebben binnen de querytaal. De uitleg van de verschillende queries is aan bod gekomen in hoofdstuk 5 tot en met hoofdstuk 7. Er blijven ook nog een aantal queries over die geen equivalent hebben in de andere querytalen.

Tabel 10: Vergelijking queries

EPL	**StreamSQL**	**CCL**
-	CREATE INPUT STREAM	CREATE INPUT STREAM
-	CREATE OUTPUT STREAM	CREATE OUTPUT STREAM
-	CREATE STREAM	CREATE LOCAL STREAM
-	CREATE SCHEMA	CREATE SCHEMA
-	CREATE WINDOW	CREATE WINDOW
RETAIN	SIZE	KEEP
-	DECLARE Variable	CREATE VARIABLE
-	APPLY JAVA	ATTACH ADAPTER
SELECT FROM	SELECT FROM	SELECT FROM
MATCHING	FROM PATTERN	MATCHING
WHERE	WHERE	WHERE
HAVING	HAVING	HAVING
GROUP BY	GROUP BY	GROUP BY
ORDER BY	ORDER BY	ORDER BY
INSERT INTO	INTO (Stroom, window), INSERT INTO (Tabel)	INSERT INTO

OUTPUT	-	OUTPUT
-	LIMIT	LIMIT
-	DELETE FROM	DELETE FROM
-	UPDATE	UPDATE
-	MERGE, UNION, GATHER	MERGE

Bij de vergelijking van EPL, StreamSQL en CCL valt het onmiddellijk op dat bij EPL geen DDL-commando's aanwezig zijn. Verder komen de drie querytalen op het vlak van structuur, conventies en commando's wel grotendeels overeen. Er zijn natuurlijk wel enkele kleine verschillen. Enkele voorbeelden:

De resultaten van een bepaalde query moeten terecht komen in een stroom, window of tabel.Het commando dat dit verwezelijkt verschilt licht tussen de drie querytalen. Bij EPL en CCL wordt het commando 'INSERT INTO' gebruikt voor zowel een stroom, een window als een tabel. Bij StreamSQL wordt 'INSERT INTO' voor een tabel gebruikt, terwijl het commando 'INTO' gebruikt wordt voor een stroom of een tabel. 'INSERT INTO' wordt bovenaan een query geplaatst, terwijl INTO onderaan een query wordt geplaatst.

Het creëren van een stroom gebeurt ongeveer hetzelfde bij StreamSQL en CCL. Enkel het definiëren van het schema gebeurt anders. Terwijl bij StreamSQL het schema tussen haakjes achter de naam van de stroom wordt vastgelegd, moet bij CCL expliciet het woord 'SCHEMA' gebruikt worden. Ook de namen van de datatypes van de velden verschillen soms (int <-> integer).

Het invoegen van een adapter is wel heel verschillend bij StreamSQL en CCL. Terwijl bij CCL deze adapter expliciet in een query moet gedefinieerd worden met behulp van 'ATTACH INPUT ADAPTER', kan dit bij StreamBase op twee verschillende manieren gebeuren. Met behulp van de 'feed simulation' gebeurt dit extern van de queries terwijl het 'APPLY'-commando gebruikt kan worden om een 'embedded adapter' toe te voegen. In de casestudie wordt gekozen om de feed simulation te gebruiken vermits het APPLY-commando redelijk omslachtig is. Bij beide platformen is het wel erg belangrijk dat de adapter in het juiste formaat staat, in dit geval een CSV-bestand zodat het programma de data kan lezen. Het is belangrijk dat de namen van de velden in de adapter overeenkomen met de gedefinieerde namen van het schema in de queries. Anders worden de waarden in de adapter niet of verkeerd gelezen.

Het zijn dus geen grote verschillen, maar zijn deze queries gemakkelijk hanteerbaar in de platformen?

Een van de zaken die onmiddellijk opvalt bij het invoeren van de queries is het gebruik van verschillende kleuren om een beter overzicht te bewaren over het geheel van queries. Zowel CCL als StreamSQL maken gebruik van kleurencodes voor onder andere gereserveerde woorden en commentaar. StreamSQL houdt het wel op neutrale, donkere kleuren, terwijl CCL voor contrasterende, felle kleuren gaat. De felle kleuren van CCL zorgen voor meer duidelijkheid en onderscheid.

Tabel 11: Kleurencodes platformen

Wat?	StreamSQL	CCL
Namen van kolommen,stromen, windows,…	Zwart	Zwart
Gereserveerde woorden	Aubergine	Licht blauw
'String'	Blauw	Roos
"String"	Blauw	Rood
Commentaar	Donker groen	Licht groen

Een tweede belangrijke punt is de autocompletion-functie van beide platformen. Zodra de gebruiker een woord begint te typen in Coral8, komt er een lijst met de verschillende opties van gereserveerde woorden of reeds gebruikte woorden in het project te voorschijn. In tegenstelling tot Coral8, komt deze lijst niet automatisch bij StreamBase. Door ctrl+space in te duwen als je met een woord bezig bent, komt wel een veel uitgebreidere lijst te voorschijn dan bij Coral8. Bij Coral8 beperkt de autocompletion zich tot het woord, terwijl bij StreamSQL alle mogelijke opties van een commando beschreven worden. Verder is het zo dat indien een gebruiker de muis op een query plaatst, er een volledige uitleg over de query verschijnt bij StreamSQL. Dit is natuurlijk erg handig indien een gebruiker niet meer weet wat er juist allemaal in de query moet staan. Bij CCL moet er dus sneller naar een hulpbron gegrepen worden.

Bij het ingeven van queries gebeurt het natuurlijk regelmatig dat er fouten gemaakt worden. Het vergeten afsluiten van een query, een typefout of een verkeerde conventie zijn maar enkele voorbeelden. Bij StreamBase worden fouten onmiddellijk weergegeven in de 'typecheck errors view' zodat de gebruiker onmiddellijk weet wanneer hij/zij een fout maakt. Het type fout en de regel waarop de fout gemaakt wordt, worden onmiddellijk aangeduid. Bij Coral8 gebeurt dit echter niet. Hier kan men wel aan de 'flow view' zien dat er fouten in de queries zitten doordat de event flow simpelweg niet wordt weergegeven indien er fouten in de queries staan. 'Flow view disabled because of compile errors' verschijnt dan onder dit scherm. Indien de gebruiker zelf de fout niet vindt of wanneer het gaat om een heel groot project, kan hij/zij het project proberen te laten uitvoeren om te weten te komen welke fouten op welke regel gemaakt zijn. Deze verschijnen dan in de 'output view'. Dit is natuurlijk ingewikkelder dan bij StreamBase. Indien het om een kleine schrijffout gaat, bijvoorbeeld het definiëren van een inputstroom met

een hoofdletter en dan daarna verwijzen naar deze inputstroom zonder hoofdletter, wordt dit bij CCL automatisch aangepast terwijl dit bij StreamBase als een fout geregistreerd wordt.

8.3 Casestudie 'verplaatsingsgedrag van gezinsleden'

De casestudie 'verplaatsingsgedrag van gezinsleden' gaat gebruikt worden om de continue querytalen van StreamBase en Coral8 te testen en om de vergelijking tussen de platformen verder te zetten. Het doel van deze casestudie is te kijken of deze queries gemakkelijk hanteerbaar zijn en efficiënt werken. Na de theoretische benadering, wordt er hier gekeken hoe het in de praktijk zou werken. Voor de dataset van dit onderzoek wordt terugverwezen naar hoofdstuk 4.

De projecten die nu volgen worden telkens in Coral8 en StreamBase aangemaakt. Op deze manier kunnen de gebruikte queries gemakkelijk vergeleken worden. De queries worden in de kleuren weergegeven zoals ze in het platform verschijnen. De groene tekst wordt zowel bij Coral8 als bij Streambase om commentaar weer te geven. In deze commentaar wordt telkens verduidelijkt wat bepaalde queries doen. Hier gaat ook onmiddellijk duidelijk worden of het effectief zo is dat een gebruiker die SQL kent ook gemakkelijk de continue querytalen kan hanteren.

Alvorens de casestudie wordt aangevangen, een kleine opmerking: de resultaten van de queries die worden weergegeven zijn niet 100% betrouwbaar vermits het over een kleine dataset gaat waarbij heel wat gegevens ontbreken (bijvoorbeeld gezinnen met kinderen onder de 6 jaar en boven de 12 jaar). In de praktijk zijn de gegevens ook van een hele andere aard dan de gegevens die nu gebruikt gaan worden. De volgende projecten dienen dan ook louter ter illustratie om te laten zien wat er bereikt kan worden met de queries en hoe de resultaten geïnterpreteerd kunnen worden.

8.3.1 Verplaatsing kinderen met de auto

Zowel ouders als kinderen zijn vaak afhankelijk van elkaar als het over verplaatsingen tussen thuis, het werk en school gaat. Meestal vindt men kinderen te jong om zich alleen te verplaatsen en zorgen ouders voor begeleiding van en naar school. Het eerste middelbaar is in vele gevallen het keerpunt waarop kinderen zelfstandig naar school gaan (Hannes, et al., 2010).

Het eerste project, 'ChildCar' genaamd, gaat trachten te onderzoeken of er een verband is tussen de leeftijd van kinderen en hoe kinderen zich verplaatsen naar school. Doordat de

dataset enkel kinderen met een leeftijd van 6 tot en met 13 jaar, wordt met de volgende queries onderzocht of dat deze evolutie eventueel al eerder zichtbaar is. Daarom wordt er een onderscheid gemaakt tussen kinderen tot en met negen jaar en boven negen jaar.

8.3.1.1 Coral8

```
-- Schema creëren voor de stromen
CREATE SCHEMA TravelSchema (
HouseholdID         LONG,     PersonID          STRING,   PersonAge          INTEGER,
Sex                 INTEGER,  HouseholdType     INTEGER,  TravelActivity     STRING,
TravelTime          INTEGER,  Mode              INTEGER,  ChildTransportation INTEGER,
Cars                INTEGER,  Morning           STRING,   WorkStatus         INTEGER
);

-- Inputstroom creëren
CREATE INPUT STREAM TravelBehaviour
SCHEMA TravelSchema;

--Adapter aan de inputstroom linken
ATTACH INPUT ADAPTER Travel TYPE ReadFromCsvFileAdapterType
TO STREAM TravelBehaviour
PROPERTIES
   FILENAME          = "$ProjectFolder\data\CSV data travelbehaviour.csv",
   RATE               = "10",
   USECURRENTTIMESTAMP = "true",
   TIMESTAMPCOLUMN       = "false";

-- Lokale stromen en outputstroom creëren
CREATE LOCAL STREAM Kind
SCHEMA TravelSchema;

CREATE LOCAL STREAM Auto
SCHEMA TravelSchema;

CREATE OUTPUT STREAM KindFiltered
SCHEMA TravelSchema;

-- Filters instellen
INSERT INTO Kind
SELECT * FROM TravelBehaviour
WHERE Activity = 'S';

INSERT INTO Auto
SELECT * FROM KIND
WHERE Mode = 5;

INSERT INTO FilteredKind
SELECT * FROM Auto
WHERE PersonAge <=9;
```

Vermits dit het eerste volledige project is, worden de queries eerst kort en bondig uitgelegd. Eerst en vooral wordt het schema, 'TravelSchema' aangemaakt waarin de velden van elk event gedefinieerd worden. Dit is meestal voor elk project de eerste stap. Elk veld krijgt een naam en

een datatype toegewezen. Om het eenvoudig te houden, worden gewoon de namen van de verschillende variabelen gebruikt als veldnamen. Vervolgens wordt een inputstroom 'TravelBehaviour' aangemaakt. Deze inputstroom maakt gebruik van het schema 'TravelSchema'. Om de data van het CSV-bestand naar deze inputstroom te zenden, wordt er daarna een inputadapter 'Travel' gedefinieerd van het type 'ReadFromCsvFileAdapterType'. De snelheid waarmee de events arriveren, wordt ook vastgelegd in de eigenschappen: 'rate' = 10. Dit wil zeggen dat er per seconde tien events in de inputstroom terecht komen. In de eigenschappen van deze adapter moet ook het bestand, waarin de data staat, aangegeven worden. Een andere belangrijke eigenschap van de inputadapter is de timestamp. Voor dit project wordt gewoon het huidige tijdstip gebruikt als timestamp: 'usecurrentimestamp' = true. Dit wil dus zeggen dat er vanuit gegaan wordt dat er geen kolom met een timestamp bestaat in dit bestand: 'Timestampcolumn' = false. Deze eigenschappen kunnen door de gebruiker eenvoudig aangevinkt worden in de 'properties view'. Dit scherm verschijnt automatisch zodra deze query wordt aangemaakt. De query wordt dan op zijn beurt automatisch aangepast aan de vereisten van de gebruiker.

Indien de gebruiker nu het project zou starten, is er al een inkomende stroom van data te zien op het scherm. Vanaf nu kunnen er queries op deze inputstroom uitgevoerd worden om betekenisvolle events uit deze stroom te bekomen. Een belangrijke opmerking: elke query moet worden afgesloten met ; dit wordt namelijk snel vergeten.

Om op het einde een goed overzicht te kunnen bewaren, worden er aansluitend twee lokale stromen gecreëerd: 'Kind' en 'Auto'. Beide lokale stromen krijgen het schema 'TravelSchema' mee. Deze stromen worden aangemaakt daar deze de analyse van de statistieken vergemakkelijken wanneer het project wordt uitgevoerd. Daarenboven wordt er een outputstroom 'FilteredKind' aangemaakt waar uiteindelijk de gefilterde events in terecht komen.

In de eerste lokale stroom komen dan alle events van kinderen van de inputstroom door de volgende voorwaarde te stellen in het 'WHERE'-commando: 'Activity' = 'S'. Vervolgens worden alle events van de kinderen gefilterd op de wijze van hun verplaatsing: 'Mode' = 5. Alle kinderen die met de auto naar school worden gebracht, komen op deze manier terecht in de tweede lokale stroom 'Auto'. Ten slotte wordt deze laatste lokale stroom gefilterd op de leeftijd van de kinderen. Kinderen jonger dan of juist 9 jaar oud worden uiteindelijk in de outputstroom weergegeven. Door eerst de twee lokale stromen te maken, kunnen nu heel eenvoudig de statistieken vergeleken worden in de 'event view'. In de 'stream viewer' worden alle stromen weergegeven zodra het project uitgevoerd wordt.

Om een beeld te krijgen wat deze queries nu exact doen, volgen hier de resultaten na het uitvoeren van het project. Van de 2396 events zijn er 984 die voldoen aan de voorwaarde van

de eerste lokale stroom 'Kind'. Dat wil zeggen dat er 984 keer een verplaatsing plaatsvindt van kinderen op deze dag. Van deze 984 verplaatsingen door kinderen zijn er 644 events die dan weer voldoen aan de voorwaarde van de tweede lokale stroom 'Auto'. Deze kinderen worden dus allemaal met de auto naar school gebracht. In de outputstroom komen uiteindelijk 452 events terecht. Dat wil zeggen dat 70,2% van de kinderen die met de auto naar school gebracht worden jonger of juist negen jaar oud zijn. Dit kan een voorteken zijn van de vooraf gestelde evolutie dat het eerste middelbaar een keerpunt is voor de meeste kinderen om zelfstandig naar school te gaan. In bijlage 13 is een screenshot te vinden van de 'flow view' waarin deze aantallen zijn weergegeven (zie 'in' en 'out') samen met de 'stream viewer' van dit project.

Hier kan wel opgemerkt worden dat deze cijfers eigenlijk niet representatief zijn aangezien het aantal kinderen per leeftijd niet bekend is. Indien de dataset van naderbij bekeken wordt, valt onmiddellijk op dat er veel meer gegevens beschikbaar zijn van kinderen tussen 6 en 9 jaar dan kinderen boven negen jaar oud. Het is dan ook eerder de bedoeling om verder te werken met de data die uit de applicatie voortkomt. De volgende uitbreidingen kunnen daarom zeker de effectiviteit van het project vergroten.

- In plaats van met één outputstroom te werken, kan er een tweede outputstroom toegevoegd worden met data van kinderen boven de negen jaar. Zo worden deze events niet weggefilterd, maar juist opgevangen in de tweede datastroom. De twee outputstromen kunnen dan door externe applicaties met elkaar vergeleken worden.

  Voorbeeld:

```
-- Tweede outputstroom creëren voor kinderen boven negen jaar
CREATE OUTPUT STREAM KindFiltered2
SCHEMA TravelSchema;

INSERT INTO KindFiltered2
SELECT * FROM Auto
WHERE PersonAge > 9;
```

  Indien nu de queries worden uitgevoerd zien we dat er slechts 192 events beschikbaar zijn van kinderen boven de negen jaar ten opzichte van 452 events van kinderen onder negen jaar. Er is dus inderdaad veel minder data beschikbaar van kinderen tussen 9 en 13 jaar.

- Stel dat er nu elke dag gegevens binnenkomen van gezinnen met kinderen van 0 tot studenten van 23 jaar en men wil de leeftijd waarop gefilterd wordt, kunnen veranderen zodat er meerdere analyses mogelijk zijn. Dit kan door een bijkomende stroom en een variabele ('CREATE VARIABLE') die deze leeftijd bevat, te creëren. De gebruiker kan dan eenvoudig via deze stroom de gewenste leeftijd ingeven en in de query wordt dan niet

de waarde van de leeftijd gebruikt, maar de naam van de nieuwe variabele die de waarde van de leeftijd bevat.

Voorbeeld:

```
-- Regelbare variabele creëren en in de queries verwerken
CREATE VARIABLE INTEGER LeeftijdFilter;
CREATE STREAM LeeftijdIngeven SCHEMA (NieuweLeeftijd INTEGER);
ON LeeftijdIngeven
SET LeeftijdFilter = LeeftijdIngeven.NieuweLeeftijd;

INSERT INTO FilteredKind
SELECT * FROM Auto
WHERE PersonAge <= LeeftijdFilter;

INSERT INTO FilteredKind2
SELECT * FROM Auto
WHERE PersonAge > LeeftijdFilter;
```

- De gegevens die uit deze queries voortkomen, worden meestal door een externe applicatie gebruikt. De events van de outputstroom moeten daarom terug in het juiste formaat getransformeerd worden zodat deze applicatie de gegevens kan lezen en gebruiken. Dit gebeurt door een outputadapter achter de outputstroom te plaatsen. Deze outputadapter vangt dan de events van de outputstroom op.

Voorbeeld:

```
-- Outputadapter linken aan de outputstroom KindFiltered
ATTACH OUTPUT ADAPTER Outputadapter TYPE WriteToCsvFileAdapterType
TO STREAM FilteredKind
PROPERTIES
   FILENAME          = "$ProjectFolder\data\outputadapter.csv",
   USECURRENTTIMESTAMP = "true"
;
```

8.3.1.2 StreamBase

Hetzelfde project 'ChildCar' wordt ook aangemaakt in StreamBase. Hier wordt geen schema vooraf gedefinieerd zodat voor elke stroom een aantal velden apart bepaald kunnen worden. Op deze manier worden de velden van de resulterende events beperkt tot diegene die belangrijk zijn. De twee mogelijkheden (8.3.1.1 en 8.3.1.2) kunnen zowel bij StreamBase als bij Coral8 gebruikt worden. Dit levert de volgende queries op.

```
-- Inputstroom creëren
CREATE INPUT STREAM TravelBehaviour(
HouseholdID LONG, PersonID INT, Activity STRING, Age INT, Sex INT, HouseholdType INT,
TravelActivity STRING, Traveltime INT, Mode INT, ChildTransportation INT, Cars INT, Morning
STRING, Workstatus INT
);
```

```
-- Stromen creëren voor kinderen, kinderen met de auto en de uiteindelijke outputstroom
CREATE STREAM Kind(
HouseholdID LONG, PersonID INT, Age INT, HouseholdType INT, TravelActivity STRING,
Traveltime INT, Mode INT);

CREATE STREAM Auto(
HouseholdID LONG, PersonID INT, Age INT, HouseholdType INT, TravelActivity STRING,
Traveltime INT);

CREATE OUTPUT STREAM KindFiltered(
HouseholdID LONG, PersonID INT, Age INT, HouseholdType INT, TravelActivity STRING,
Traveltime INT);

-- Events selecteren van kinderen
SELECT HouseholdID, PersonID, Age, HouseholdType, TravelActivity, Traveltime, Mode
FROM TravelBehaviour
WHERE Activity = 'S'
INTO Kind;

-- Events selecteren van kinderen die met de auto naar school gaan
SELECT HouseholdID, PersonID, Age, HouseholdType, TravelActivity, Traveltime
FROM Kind
WHERE Mode = 5
INTO Auto;

-- Events selecteren van kinderen jonger dan of juist negen jaar oud
SELECT HouseholdID, PersonID, Age, HouseholdType, TravelActivity, Traveltime
FROM Auto
WHERE Age <=9
INTO KindFiltered;
```

In bijlage 14 zijn de resultaten van deze queries te vinden in de ‘application view’ en de ‘StreamBase monitor’. Deze zijn helemaal niet zo uitgebreid als bij Coral8. De events in de verschillende stromen worden weergegeven in tabbladen, dus input en output kunnen niet samen getoond worden in de ‘application view’. Als er een StreamSQL project aangemaakt wordt, is het daarnaast zo dat er geen ‘event flow’ wordt weergegeven zoals bij Coral8. Er kunnen dus ook niet zomaar statistieken geëvalueerd worden. In de ‘StreamBase monitor’ kunnen er beperkt statistieken geraadpleegd worden. De ‘streams view’ is te vinden in bijlage 14. Onderaan deze figuur is te zien dat er 452 events in de outputstroom terecht zijn gekomen (default.KindFiltered). Het uiteindelijke resultaat is hetzelfde als bij Coral8, maar de statistieken van de lokale stromen worden niet weergegeven. Er kunnen dus geen vergelijkingen gemaakt worden tussen de verschillende stromen. Meestal is het toch de bedoeling dat de betekenisvolle events van de outputstroom gebruikt worden voor verdere analyse door externe applicaties.

Doordat de applicatie dus zelf weinig vertelt over het uitgevoerd project, is het zeker belangrijk dat er een outputadapter wordt toegevoegd zodat de resultaten gebruikt kunnen worden door een externe applicatie. Met behulp van het ‘APPLY’-commando kan er een ‘CSV Writer’-adapter worden toegevoegd zodat de events van de outputstroom hierin terecht komen.

Voorbeeld:

```
-- Outputadapter creëren om events op te vangen
APPLY JAVA "com.streambase.sb.adapter.csvwriter.CSVWriter" AS OutputAdapterTravel
(IncludeHeaderInFile = "true", MaxRollSecs = "0", StringQuoteOption = "Quote if necessary",
FlushInterval = "1", StringQuoteCharacter = "\"", RollPeriod = "None", FieldDelimiter = ",",
TsFormat = "false", CompressData = "false", MaxFileSize = "0",OpenOutputFileDuringInit =
"false", ThrottleErrorMessages = "false", NullValueRepresentation = "null", FileName =
"outputstroom.csv", IfFileDoesntExist = "Create new file", CheckForRollAtStartup = "false",
SyncOnFlush = "false", IfFileExists = "Append to existing file")
FROM KindFiltered;
```

Ook hier kan weer een variabele aangemaakt worden om de filter op de leeftijd te regelen in plaats van een getal in de queries te gebruiken. Dit gebeurt bij StreamSQL met het 'DECLARE'-commando. Aan deze variabele kan een beginwaarde gegeven worden met behulp van 'DEFAULT' (9 jaar) zodat het oorspronkelijke project wordt uitgevoerd.

Voorbeeld:

```
-- Dynamische variabele creëren
CREATE INPUT STREAM LeeftijdIngeven(NieuweLeeftijd INT);
DECLARE LeeftijdFilter INT DEFAULT 9
UPDATE FROM (SELECT LeeftijdIngeven.NieuweLeeftijd FROM LeeftijdIngeven);

-- Nieuwe variabele gebruiken in een query
SELECT HouseholdID, PersonID, Age, HouseholdType, TravelActivity, Traveltime
FROM Auto
WHERE Age <= LeeftijdFilter
INTO KindFiltered;
```

8.3.2 Verplaatsingen kinderen door vrouwen

In heel wat gezinnen komt het vaak voor dat vrouwen hun werk combineren met de verantwoordelijkheid over de kinderen. In tweeoudergezinnen, waarbij de twee ouders werken, komt het daarnaast vaak voor dat de vrouw parttime werkt om dit te kunnen combineren. Indien de vrouw toch fulltime werkt, gebeurt het dikwijls dat de kinderzorg gedelegeerd wordt door de vrouw. Bij tweeoudergezinnen met één kostwinner is het ook meestal de vrouw die niet werkt en dus de kinderzorg volledig op haar schouders neemt (Hannes, et al., 2010).

In dit project 'ChildrenTransport' gaat geanalyseerd worden welke activiteiten (dropoff, pickup, dropoff&pickup) door de vrouw in het gezin gedaan worden en wat de werkstatus is van deze vrouwen. De volgende queries gaan de verschillende activiteiten (dropoff, pickup, dropoff&pickup) in verschillende stromen indelen. Vervolgens wordt er gekeken in hoeveel events de vrouw deze verantwoordelijkheid op zich neemt en wordt er gekeken of deze vrouwen fulltime, parttime werken of helemaal niet werken. Uiteindelijk komen er drie outputstromen uit de applicatie met enkel events van vrouwen die zich bezig houden met het transport van de kinderen gesplitst op basis van hun werkstatus.

8.3.2.1 Coral8

```
-- Inputstroom creëren
CREATE INPUT STREAM TravelBehaviour
SCHEMA(
HouseholdID LONG, PersonID INTEGER, Activity STRING, PersonAge INTEGER, Sex INTEGER,
HouseholdType INTEGER, TravelActivity STRING, Traveltime INTEGER, Mode INTEGER,
ChildTransportation INTEGER, Cars INTEGER, Morning STRING, WorkStatus INTEGER
);

--Adapter linken aan de inputstroom
ATTACH INPUT ADAPTER Travel TYPE ReadFromCsvFileAdapterType
TO STREAM TravelBehaviour
PROPERTIES
   FILENAME          = "$ProjectFolder\data\CSV data travelbehaviour.csv",
   RATE              = "5",
   TIMESTAMPCOLUMN     = "false",
   USECURRENTTIMESTAMP = "true"
;

--Schema aanmaken voor de locale stromen
CREATE SCHEMA stroom
(HouseholdID LONG, PersonID INTEGER, PersonAge INTEGER, Sex INTEGER, HouseholdType
INTEGER, Mode INTEGER, ChildTransportation INTEGER, WorkStatus INTEGER)
;

-- Lokale stromen creëren
CREATE LOCAL STREAM Dropoff
SCHEMA stroom;

CREATE LOCAL STREAM Pickup
SCHEMA stroom;

CREATE LOCAL STREAM DropoffAndPickup
SCHEMA stroom;

--Eenvoudige manier om een stroom op te splitsen in meerdere stromen
INSERT
  WHEN ChildTransportation = 1 THEN Dropoff
  WHEN ChildTransportation = 2 THEN Pickup
  WHEN ChildTransportation = 3 THEN DropoffAndPickup
SELECT HouseholdID, PersonID, PersonAge, Sex, HouseholdType, Mode, ChildTransportation,
Workstatus
FROM TravelBehaviour;

-- Stroom creëren
CREATE LOCAL STREAM Vrouw
SCHEMA stroom;

-- Events selecteren waarbij de vrouw de kinderen begeleid naar school
INSERT INTO Vrouw
SELECT * FROM Dropoff WHERE Sex = 2;
INSERT INTO Vrouw
SELECT * FROM Pickup WHERE Sex = 2;
INSERT INTO Vrouw
SELECT * FROM DropoffAndPickup WHERE Sex = 2;

-- Outputstromen creëren voor fulltime werken, parttime werken of niet werken
CREATE OUTPUT STREAM FullTime
```

```
SCHEMA stroom;

CREATE OUTPUT STREAM Parttime
SCHEMA stroom;

CREATE OUTPUT STREAM NietWerken
SCHEMA stroom;

-- Vrouwen indelen op basis van hun werkstatus
INSERT INTO FullTime
SELECT * FROM Vrouw
WHERE WorkStatus = 2;

INSERT INTO Parttime
SELECT * FROM Vrouw
WHERE WorkStatus = 1;

INSERT INTO NietWerken
SELECT * FROM Vrouw
WHERE WorkStatus = 0;
```

Het enige nieuwe element bij deze queries is het moment dat de inputstroom opgesplitst wordt in drie stromen. Dit is ook mogelijk door telkens voor elk van de drie stromen een filterquery aan te maken, maar hier zijn de drie filter queries gecombineerd in één query. Dit kan met behulp van het 'INSERT WHEN'-commando.

Uit de resultaten van het project kunnen een aantal conclusies getrokken worden. Zie bijlage 15 voor een screenshot van de flow view en de stream viewer. In 332 van de 392 gevallen zorgt de vrouw zowel voor het afzetten als het ophalen van de kinderen. Bij het afzonderlijk afzetten en ophalen van de kinderen is het opmerkelijk dat de mannen zich vooral bezig houden met het afzetten van de kinderen aan school (116 van de 200), terwijl het ophalen van de kinderen meer door de vrouw gebeurt (100 van de 152). Deze activiteiten komen vreemd genoeg niet overeen in aantallen (200<-> 152). Hier wordt verder op in gegaan in het derde project (8.3.3). Ten slotte wordt vastgesteld dat ongeveer de helft van de vrouwen, die zich bezig houden met het transport van de kinderen, parttime of helemaal niet werkt.

Een mogelijke uitbreiding zou weer zijn om een output adapter te linken aan elke outputstroom zodat deze gegevens gebruikt kunnen worden voor analyse. Daarnaast kunnen er nog verder stromen aangemaakt worden om te analyseren om wat voor soort gezin het gaat.

8.3.2.2 StreamBase

```
-- Inputstroom creëren
CREATE INPUT STREAM TravelBehaviour(
HouseholdID LONG, PersonID INT, Activity STRING, PersonAge INT, Sex INT, HouseholdType INT, TravelActivity STRING, Traveltime INT, Mode INT, ChildTransportation INT, Cars INT, Morning STRING, WorkStatus INT);
```

```
-- Schema creëren voor de stromen
CREATE SCHEMA stroom
(HouseholdID LONG, PersonID INT, PersonAge INT, Sex INT, HouseholdType INT, Mode INT,
ChildTransportation INT, WorkStatus INT);

-- Stroom aanmaken voor ouders die zich bezig houden met het transport van de kinderen en
events selecteren voor deze stroom
CREATE STREAM DropOffPickUp stroom;

SELECT HouseholdID, PersonID, PersonAge, Sex, HouseholdType, Mode, ChildTransportation,
WorkStatus
FROM TravelBehaviour
WHERE NOT (ChildTransportation = 4 OR ChildTransportation = 5) INTO DropOffPickUp;

-- Events selecteren waarbij de vrouw zich bezig houdt met het transport van de kinderen
CREATE STREAM Vrouw stroom;

SELECT HouseholdID, PersonID, PersonAge, Sex, HouseholdType, Mode, ChildTransportation,
WorkStatus
FROM DropOffPickUp
WHERE Sex = 2 INTO Vrouw;

-- Outputstromen creëren met de verschillende werkstatussen
CREATE OUTPUT STREAM FullTime stroom;
CREATE OUTPUT STREAM PartTime stroom;
CREATE OUTPUT STREAM NietWerken stroom;

-- Vrouwen indelen naargelang hun werkstatus
SELECT HouseholdID, PersonID, PersonAge, Sex, HouseholdType, Mode, ChildTransportation,
WorkStatus
FROM Vrouw
WHERE WorkStatus = 2 INTO FullTime;

SELECT HouseholdID, PersonID, PersonAge, Sex, HouseholdType, Mode, ChildTransportation,
WorkStatus
FROM Vrouw
WHERE WorkStatus = 1 INTO PartTime;

SELECT HouseholdID, PersonID, PersonAge, Sex, HouseholdType, Mode, ChildTransportation,
WorkStatus
FROM Vrouw
WHERE WorkStatus = 0 INTO NietWerken;
```

Aangezien de lokale stromen bij StreamBase niet van belang zijn (zie vorige case) wordt er hier geen opdeling gemaakt naar dropoff, pickup en dropoff&pickup. De events die er niet toe doen worden gewoon weggefilterd (NOT(ChildTransportation = 4 OR ChildTransportation = 5)). Op deze manier blijven enkel de events over waarbij de ouders zich bezig houden met het transport van de kinderen. Verder lopen de queries gelijkaardig aan die van Coral8. Bijlage 16 toont een voorbeeld van de 'application view' van de outputstroom 'FullTime' en de 'StreamBase monitor' na het uitvoeren van het project. Ook hier zijn enkel het aantal events af te lezen die in de outputstromen terecht komen. Een outputadpater kan weer toegevoegd worden op dezelfde manier als in het eerste project.

8.3.3 Niet plaatsvinden van events – Coral8

In het vorige project 'ChildTransport' werd reeds opgemerkt dat het aantal activiteiten 'dropoff' en 'pickup' niet overeenkomen. Normaal zou men toch veronderstellen dat indien een kind naar school gebracht wordt, dat dit kind ook terug opgehaald wordt door de ouders of andersom. Het kan dus blijkbaar voorvallen dat deze activiteiten niet overeenkomen en dat er dus bepaalde events niet plaatsvinden. Het gebeurt regelmatig dat een gebruiker ook geïnteresseerd is in events die niet hebben plaatsgevonden en die normaal wel moesten plaatsvinden, zogenaamde non-events. Met dit project 'DropOffPickup' wordt er nagegaan of er gevallen zijn waarbij dit effectief zo is.

Er wordt dus eigenlijk gezocht naar een patroon waarbij één event wel plaatsvindt (bijvoorbeeld het kind naar school brengen) en waarbij het tweede event niet plaatsvindt (bijvoorbeeld het niet afhalen van een kind aan school). Hier zijn meerdere verklaringen voor. Een mogelijke verklaring zou kunnen zijn dat er externe personen worden ingeschakeld om kinderen naar school te brengen of van school terug naar huis te brengen. Een andere mogelijkheid zou kunnen zijn dat kinderen één van de twee verplaatsingen zelfstandig maken. In het vierde project wordt dit project nog verder uitgebreid om te kijken in welke gevallen er echt een extern persoon wordt ingezet.

8.3.3.1 Coral8

De volgende queries werden aangemaakt:

```
-- Inputstroom voor mannen creëren
CREATE INPUT STREAM Man
SCHEMA (HouseholdID LONG, PersonID INTEGER, Activity STRING, PersonAge INTEGER, Sex
INTEGER, LeaveHTime INTEGER, ArriveWSTime INTEGER, LeaveWSTime INTEGER,
ArriveHTime INTEGER, Mode INTEGER, ChildTransportation INTEGER, Cars INTEGER, Morning
STRING, Workstatus INTEGER);

-- Inputadapter aan deze stroom linken
ATTACH INPUT ADAPTER DataMan TYPE ReadFromCsvFileAdapterType
TO STREAM Man
PROPERTIES
   FILENAME          = "$ProjectFolder\data\data mannen2.csv",
   RATE              = "10",
   USECURRENTTIMESTAMP = "true",
   TIMESTAMPCOLUMN     = "false";

-- Inputstroom voor vrouwen creëren
CREATE INPUT STREAM Vrouw
SCHEMA(HouseholdID LONG, PersonID INTEGER, Activity STRING, PersonAge INTEGER, Sex
INTEGER, LeaveHTime INTEGER, ArriveWSTime INTEGER, LeaveWSTime INTEGER, ArriveHTime
INTEGER, Mode INTEGER, ChildTransportation INTEGER, Cars INTEGER, Morning STRING,
Workstatus INTEGER);

-- Inputadapter aan deze stroom linken
ATTACH INPUT ADAPTER DataVrouw TYPE ReadFromCsvFileAdapterType
```

```
TO STREAM Vrouw
PROPERTIES
   FILENAME          = "$ProjectFolder\data\data vrouwen2.csv",
   RATE              = "10",
   USECURRENTTIMESTAMP = "true",
   TIMESTAMPCOLUMN     = "false";

-- Outputstroom creëren
CREATE OUTPUT STREAM StreamAlert
SCHEMA(Alert STRING, HouseholdID LONG, ManChildTransportation INTEGER,
VrouwChildTransportation INTEGER);

-- Zoeken naar events die niet hebben plaatsgevonden: Mismatch tussen het transport van de
kinderen
INSERT INTO StreamAlert
SELECT "Mismatch", Man.HouseholdID, Man.ChildTransportation, Vrouw.ChildTransportation
FROM Man, Vrouw
MATCHING [20 SECONDS: Man, Vrouw]
ON Man.HouseholdID = Vrouw.HouseholdID
WHERE (Man.ChildTransportation = 1 OR Man.ChildTransportation = 2) AND
Vrouw.ChildTransportation = 4;

INSERT INTO StreamAlert
SELECT "Mismatch", Man.HouseholdID, Man.ChildTransportation, Vrouw.ChildTransportation
FROM Man, Vrouw
MATCHING [20 SECONDS: Vrouw, Man]
ON Man.HouseholdID = Vrouw.HouseholdID
WHERE Man.ChildTransportation = 4 AND (Vrouw.ChildTransportation = 1 OR
 Vrouw.ChildTransportation = 2);
```

Het eerste wat opvalt, is dat er met twee inputstromen gewerkt wordt. Dit is noodzakelijk om in de volgende queries te kunnen zoeken naar patronen met behulp van 'MATCHING'. De events zijn dus voor dit project opgesplitst in 2 inputstromen: 'man' en 'vrouw'. De events van kinderen worden in dit project weggelaten.

Om een bepaald patroon te vinden wordt dus gebruik gemaakt van 'MATCHING'. Hier is gekozen voor een willekeurig aantal seconden om te kijken of een event van de ene stroom gevolgd wordt door een event van de andere stroom waarbij het 'householdID' van de events overeenkomt. Er wordt dus gezocht naar een patroon binnen een gezin. De voorwaarden voor deze events worden in het 'WHERE'-commando geformuleerd. Een van de twee ouders brengt of haalt hun kinderen van school (ChildTransportation = 1 OR ChildTransportation = 2), terwijl de andere ouder geen van de twee activiteiten uitvoert (ChildTransportation = 4). De events waarbij dit patroon voorkomt, worden naar de outputstroom 'StreamAlert' gezonden. De eerste kolom van deze outputstroom heeft als naam 'Alert' en hierin verschijnt het woord 'Mismatch' telkens wanneer er een gezin is waarbij de activiteiten niet overeenkomen. De overige kolommen bevatten de 'householdID' van het gezin en de 'ChildTransportation' activiteiten van de twee ouders.

In bijlage 17 zijn de resultaten van deze queries te vinden in de 'event flow' en de 'stream viewer' van de outputstroom. Er zijn dus 26 gezinnen op een totaal van 500 waarbij de 'dropoff' en 'pickup' activiteiten niet overeenkomen. Bij deze gezinnen is het dus mogelijk dat er een extern persoon wordt ingeschakeld. Indien dit project nu elke dag opnieuw uitgevoerd wordt met data van dezelfde gezinnen, kan er gekeken worden of dit elke dag het geval is of dat dit bepaalde dagen voorvalt. Ook hier kan weer een outputadapter toegevoegd worden om de data op te vangen en te gebruiken bij een externe applicatie.

Voorbeeld:

```
-- Outputadapter toevoegen
ATTACH OUTPUT ADAPTER Alerts TYPE WriteToCsvFileAdapterType
TO STREAM StreamAlert
PROPERTIES
   FILENAME          = "$ProjectFolder\data\outputstroom.csv",
   USECURRENTTIMESTAMP = "true";
```

8.3.3.2 StreamBase

```
-- Inputstroom creëren
CREATE INPUT STREAM TravelBehaviour
(TravelTime TIMESTAMP, HouseholdID LONG, PersonID INT, Activity STRING, PersonAge INT,
Sex INT, HouseholdType INT, TravelActivity STRING, Mode INT, ChildTransportation INT, Cars
DOUBLE, Morning STRING, WorkStatus INT
);

-- Inputstroom opdelen in een stroom voor mannen en vrouwen (exclusief kinderen)
CREATE STREAM Man;
CREATE STREAM Vrouw;

SELECT * FROM TravelBehaviour
WHERE Activity = "W" AND Sex = 1 INTO Man;

SELECT * FROM TravelBehaviour
WHERE Activity = "W" AND Sex = 2 INTO Vrouw;

-- Outputstroom creëren
CREATE OUTPUT STREAM Out (HouseholdID LONG, ChildTransportationMan INT,
ChildTransportationVrouw INT);

-- Events selecteren waarbij het patroon van het transport van kinderen niet overeen komt
SELECT Man.HouseholdID, Man.ChildTransportation AS ChildTransportationMan,
Vrouw.ChildTransportation AS ChildTransportationVrouw
FROM PATTERN (Man THEN Vrouw)
WITHIN 300 ON TravelTime
WHERE  Man.HouseholdID = Vrouw.HouseholdID AND
          ((Man.ChildTransportation = 1 OR Man.ChildTransportation = 2) AND
          Vrouw.ChildTransportation = 4) OR
          ((Vrouw.ChildTransportaton = 1 OR Vrouw.ChildTransportation = 2) AND
          Man.ChildTransportation = 4)
INTO Out;
```

Hetzelfde project wordt aangemaakt in StreamBase, waarbij toch onmiddellijk een aantal verschillen met Coral8 naar boven komen. In plaats van het 'MATCHING'-commando wordt hier het 'FROM PATTERN'-commando gebruikt. De volgorde van de events wordt weergegeven tussen haakjes en het tijdsinterval wordt achter het 'WITHIN'-commando gespecifieerd, in dit geval 5 minuten. Bij 'MATCHING' wordt er achter het 'ON'-commando bepaald op basis van welk veld er gezocht gaat worden naar een patroon. Hier moet deze voorwaarde echter gewoon toegevoegd worden aan het 'WHERE'-commando.

In plaats van dat er met meerdere inputstromen wordt gewerkt zoals bij Coral8, wordt hier de inputstroom eerst opgesplitst in twee stromen, 'man' en 'vrouw'. In deze twee lokale stromen wordt er dan gezocht naar het patroon. Het is natuurlijk ook mogelijk bij StreamBase om met de twee inputstromen te werken.

In StreamBase wordt er geen gebruik gemaakt van het huidige tijdstip als timestamp, maar hier is de dataset gebruikt waarbij de variabele 'TravelTime' in een juist formaat ("2010-05-20 uu:mm:ss") is geplaatst zodat deze als timestamp gebruikt kan worden. Dit wordt gedaan om te laten zien dat het dus mogelijk is om op deze manier dezelfde resultaten te bekomen. De resultaten van dit project zijn ook niet zo eenvoudig te vergelijken, vermits er hier gewerkt wordt met het tweede scenario van de dataset en bij Coral8 met het eerste scenario. Het gevolg is dat resultaten van een bepaald gezin meerdere keren weergegeven worden. De events die in de outputstroom terecht komen zijn te vinden in bijlage 18.

8.3.4 Verder onderzoek externe personen – Coral8

In dit project 'ExternePersonen' wordt de stroom van de kinderen toegevoegd bij het vorige project. Telkens wanneer de wijze van transport van een kind gelijk is aan 5 (passagier van een auto) en 'ChildTransportation' van beide ouders gelijk is aan 4 (Brengen of halen de kinderen niet van school), wordt er verondersteld dat er een extern persoon wordt ingeschakeld om de kinderen naar school te brengen en hen terug op te halen.

8.3.4.1 Coral8

```
-- Schema creëren voor de stromen
CREATE SCHEMA Travel
(HouseholdID LONG, PersonID INTEGER, Activity STRING, PersonAge INTEGER, Sex INTEGER,
LeaveHTime INTEGER, ArriveWSTime INTEGER, LeaveWSTime INTEGER, ArriveHTime INTEGER,
Mode INTEGER, ChildTransportation INTEGER, Cars INTEGER, Morning STRING, Workstatus
INTEGER);
-- Inputstroom creëren voor events van kinderen
CREATE INPUT STREAM Kind
SCHEMA Travel;

-- Inputadapter linken aan deze inputstroom met de data van kinderen in
```

```
ATTACH INPUT ADAPTER DataKind TYPE ReadFromCsvFileAdapterType
TO STREAM Kind
PROPERTIES
   FILENAME          = "$ProjectFolder\data\data kinderen2.csv",
   RATE              = "10",
   TIMESTAMPCOLUMN     = "false",
   USECURRENTTIMESTAMP = "true"
;

-- Inputstroom creëren voor events van mannen
CREATE INPUT STREAM Man
SCHEMA Travel;

-- Inputadapter linken aan deze inputstroom met de data van mannen in
ATTACH INPUT ADAPTER DataMan TYPE ReadFromCsvFileAdapterType
TO STREAM Man
PROPERTIES
   FILENAME          = "$ProjectFolder\data\data mannen2.csv",
   TIMESTAMPCOLUMN     = "false",
   USECURRENTTIMESTAMP = "true",
   RATE              = "10"
;

-- Inputstroom creëren voor events van vrouwen
CREATE INPUT STREAM Vrouw
SCHEMA Travel;


-- Inputadapter linken aan deze inputstroom met de data van vrouwen in
ATTACH INPUT ADAPTER DataVrouw TYPE ReadFromCsvFileAdapterType
TO STREAM Vrouw
PROPERTIES
   FILENAME          = "$ProjectFolder\data\data vrouwen2.csv",
   RATE              = "10",
   USECURRENTTIMESTAMP = "true",
   TIMESTAMPCOLUMN     = "false"
;

-- Outputstroom creëren voor de aanwezigheid van externe personen
CREATE OUTPUT STREAM ExternePersonen
SCHEMA (Extern STRING, HouseholdID LONG, Mode INTEGER, ChildTransportationMan
INTEGER, ChildTransportationVrouw INTEGER);

-- Zoeken naar patronen tussen de drie inputstromen waarbij externe personen voor het
transport van de kinderen zouden zorgen zowel dropoff als pickup
INSERT INTO ExternePersonen
SELECT "Ja", Kind.HouseholdID, Kind.Mode, Man.ChildTransportation,
Vrouw.ChildTransportation
FROM Kind, Man, Vrouw
MATCHING [2 MINUTES: Man, Vrouw, Kind]
ON Kind.HouseholdID = Man.HouseholdID = Vrouw.HouseholdID
WHERE (Kind.Mode = 5 AND Man.ChildTransportation = 4 AND Vrouw.ChildTransportation = 4)
;
```

Het is belangrijk dat de juiste volgorde in het patroon in de query wordt gebruikt. Dit kan niet willekeurig bepaald worden. De datastroom van de man is veel kleiner dan de stromen van de

vrouw en de kinderen en is dan ook sneller afgelopen. Daarom moet deze voorop geplaatst worden (MATCHING [2 MINUTES: Man, Vrouw, Kind]). Stel dat bijvoorbeeld de stroom 'Kind' voorop geplaatst. Vermits deze stroom groter en dus tragre is dan de andere twee, zullen er geen patronen ontdekt worden. Deze keuze is dus erg belangrijk. In totaal zijn er 11 events waarbij het kind wel met de auto naar school gaat, maar dat de ouders deze taak niet op zich nemen.

Dit project kan nog uitgebreid worden met het vorige project. Zo kan er gezocht worden naar de events waarbij de activiteiten van de ouders om de kinderen te brengen of te halen van school niet overeen komen. De laatste query zou er dan als volgt uitzien:

Voorbeeld:

```
-- Aanwezigheid van externe personen onderzoeken ook voor dropoff of pickup
INSERT INTO ExternePersonen
SELECT "Ja", Kind.HouseholdID, Kind.Mode, Man.ChildTransportation,
Vrouw.ChildTransportation
FROM Kind, Man, Vrouw
MATCHING [2 MINUTES: Man, Vrouw, Kind]
ON Kind.HouseholdID = Man.HouseholdID = Vrouw.HouseholdID
WHERE (Kind.Mode = 5 AND Man.ChildTransportation = 4 AND Vrouw.ChildTransportation = 4)
OR (Kind.Mode = 5 AND Man.ChildTransportation = 4 AND (Vrouw.ChildTransportation = 1 OR
Vrouw.ChildTransportation = 2))
OR (Kind.Mode = 5 AND Vrouw.ChildTransportation = 4 AND (Man.ChildTransportation = 1 OR
Man.ChildTransportation = 2));
```

De resultaten van deze uitbreiding zijn te vinden in bijlage 19. In 55 gevallen is er een extern persoon dat voor het transport van een kind zorgt. Dit kan gaan over meerdere kinderen binnen één gezin. Het gaat in totaal om 41 gezinnen van 500 gezinnen. Ten slotte kan ook hier weer een outputadapter toegevoegd worden om de datastroom op te vangen.

8.3.4.2 StreamBase

```
-- Inputstroom creëren
CREATE INPUT STREAM TravelBehaviour
(TravelTime TIMESTAMP, HouseholdID LONG, PersonID INT, Activity STRING, PersonAge INT,
Sex INT, HouseholdType INT, TravelActivity STRING, Mode INT, ChildTransportation INT, Cars
DOUBLE, Morning STRING, WorkStatus INT
);

-- Inputstroom opsplitsen in drie stromen voor mannen, vrouwen en kinderen
CREATE STREAM Man;
CREATE STREAM Vrouw;
CREATE STREAM Kind;

-- Events naar deze stromen zenden
SELECT * FROM TravelBehaviour
WHERE Activity = "W" AND Sex = 1 INTO Man;
```

```
SELECT * FROM TravelBehaviour
WHERE Activity = "W" AND Sex = 2 INTO Vrouw;

SELECT * FROM TravelBehaviour
WHERE Activity = "S" INTO Kind;

-- Outputstroom creëren voor events weer te geven
CREATE OUTPUT STREAM Out (HouseholdID LONG, Mode INT, ChildTransportationMan INT,
ChildTransportationVrouw INT);

-- Zoeken naar patronen waarbij externe personen betrokken zouden zijn bij het transport van
kinderen
SELECT Man.HouseholdID, Kind.Mode, Man.ChildTransportation AS ChildTransportationMan,
Vrouw.ChildTransportation AS ChildTransportationVrouw
FROM PATTERN (Man THEN Vrouw THEN Kind)
WITHIN 300 ON TravelTime
WHERE  Man.HouseholdID = Vrouw.HouseholdID AND Man.HouseholdID = Kind.HouseholdID
        AND ((Man.ChildTransportation = 4 AND Vrouw.ChildTransportation = 4 AND Kind.Mode
        =5) OR (Kind.Mode = 5 AND Man.ChildTransportation = 4 AND
        (Vrouw.ChildTransportation = 1 OR Vrouw.ChildTransportation = 2)) OR (Kind.Mode =
         5 AND Vrouw.ChildTransportation = 4 AND (Man.ChildTransportation = 1 OR
        Man.ChildTransportation = 2)))
INTO Out;
```

In StreamBase is onmiddellijk de variant op de uitbreiding gemaakt. Met deze queries worden dus alle events, waarbij een extern persoon betrokken is, geregistreerd. Hier wordt opnieuw met één inputstroom gewerkt en met drie lokale stromen: 'man', 'vrouw' en 'kind'. In StreamBase zijn de statistieken hier niet nuttig, maar indien de gegevens opgevangen worden door een output adapter, kunnen deze events wel apart geanalyseerd worden.

8.3.5 Toepassing sliding windows

In het laatste project wordt er een toepassing gemaakt van een sliding window, 'WindowTravelBehaviour'. Het voordeel is dat op deze manier een oneindige dataset kan opgedeeld worden in kleinere delen waarop de queries worden uitgevoerd. De vertrektijden van kinderen 's ochtends naar school zouden afhankelijk zijn van de transportwijze. Er wordt verwacht dat kinderen met de auto vroeger vertrekken dan kinderen met de fiets en dat kinderen met de fiets dan weer vroeger vertrekken dan kinderen die te voet naar school gaan. 's Avonds zou dan normaal het omgekeerde effect waarneembaar moeten zijn dat kinderen die te voet zijn het eerste en kinderen met de auto het laatste thuis zijn. In het project van Coral8 gaat er per transportwijze een window aangemaakt worden waarop telkens de gemiddelde vertrektijd in de ochtend wordt berekend. Het project van StreamBase gaat hetzelfde doen maar dan voor de gemiddelde aankomsttijden 's avonds.

8.3.5.1 Coral8

```
-- Inputstroom creëren
CREATE INPUT STREAM TravelBehaviour
```

```
SCHEMA (HouseholdID LONG, PersonID INTEGER, Activity STRING, PersonAge INTEGER, Sex
INTEGER, LeaveHTime INTEGER,ArriveWSTime INTEGER, LeaveWSTime INTEGER, ArriveHTime
INTEGER, Mode INTEGER, ChildTransportation INTEGER, Cars INTEGER,
Morning STRING, WorkStatus INTEGER);

-- Inputadapter linken aan inputstroom met data van kinderen
ATTACH INPUT ADAPTER ReadFromCSVFile TYPE ReadFromCsvFileAdapterType
TO STREAM TravelBehaviour
PROPERTIES
   FILENAME          = "$ProjectFolder\data\data kinderen2.csv",
   TIMESTAMPCOLUMN     = "false",
   USECURRENTTIMESTAMP = "true",
   RATE              = "10"
;

-- Per transportwijze een window aanmaken
CREATE WINDOW WindowVoet
SCHEMA (HouseholdID LONG, PersonAge INTEGER, LeaveHTime INTEGER)
KEEP 20 ROWS;

CREATE WINDOW WindowFiets
SCHEMA (HouseholdID LONG, PersonAge INTEGER, LeaveHTime INTEGER)
KEEP 20 ROWS;

CREATE WINDOW WindowAuto
SCHEMA (HouseholdID LONG, PersonAge INTEGER, LeaveHTime INTEGER)
KEEP 20 ROWS;


-- Events toevoegen aan elk scherm afhankelijk van de transportwijze
INSERT INTO WindowVoet
SELECT HouseholdID, PersonAge, LeaveHTime FROM TravelBehaviour
WHERE Mode = 1;

INSERT INTO WindowFiets
SELECT HouseholdID, PersonAge, LeaveHTime FROM TravelBehaviour
WHERE Mode = 2;

INSERT INTO WindowAuto
SELECT HouseholdID, PersonAge, LeaveHTime FROM TravelBehaviour
WHERE Mode = 5;

-- Schema creëren voor outputstromen
CREATE SCHEMA stroom(HouseholdID LONG, PersonAge INTEGER, LeaveHTime INTEGER,
GemiddeldeVertrektijd FLOAT);

-- Outputstromen aanmaken per transportwijze
CREATE OUTPUT STREAM KindTeVoet SCHEMA stroom;
CREATE OUTPUT STREAM KindMetDeFiets SCHEMA stroom;
CREATE OUTPUT STREAM KindMetDeAuto SCHEMA stroom;

-- Events toevoegen aan outputstromen met de gemiddelde vertrektijd
INSERT INTO KindTeVoet
SELECT HouseholdID, PersonAge, LeaveHTime, AVG(LeaveHTime) FROM WindowVoet;

INSERT INTO KindMetDeFiets
SELECT HouseholdID, PersonAge, LeaveHTime, AVG(LeaveHTime) FROM WindowFiets;
```

```
INSERT INTO KindMetDeAuto
SELECT HouseholdID, PersonAge, LeaveHTime, AVG(LeaveHTime) FROM WindowAuto;
```

Nadat er een inputstroom en een inputadapter aangemaakt zijn, wordt er per transportwijze (te voet, fiets, auto) een window gecreëerd met hetzelfde schema als de inputstroom. In elk window worden er telkens 20 events weergegeven. Vervolgens wordt er per transportwijze een outputstroom aangemaakt met de gemiddelde vertrektijd. Dit gemiddelde wordt telkens opnieuw herberekend bij het binnenkomen van nieuwe events in het window per transportwijze. Telkens wanneer een nieuw event arriveert in het window, verdwijnt het oudste event uit het window. De resultaten worden in de drie 'stream viewers' weergegeven in bijlage 20. Hier is duidelijk te zien dat de gemiddelde vertrektijd van kinderen met de auto (linkse scherm) vroeger ligt dan deze met de fiets (middelste scherm) en te voet (rechtste scherm).

8.3.5.2 StreamBase

```
-- Inputstroom creëren
CREATE INPUT STREAM TravelBehaviour
(HouseholdID LONG, PersonID INT, Activity STRING, PersonAge INT, Sex INT,
LeaveHTime INT,ArriveWSTime INT, LeaveWSTime INT, ArriveHTime INT, Mode INT,
ChildTransportation INT, Cars INT,
Morning STRING, WorkStatus INT);

-- Stromen per transportwijze creëren
CREATE STREAM TeVoet (HouseholdID LONG, PersonID INT, PersonAge INT,ArriveHTime INT,
Mode INT);
CREATE STREAM Fiets (HouseholdID LONG, PersonID INT, PersonAge INT,ArriveHTime INT,
Mode INT);
CREATE STREAM Auto (HouseholdID LONG, PersonID INT, PersonAge INT,ArriveHTime INT,
Mode INT);

SELECT HouseholdID, PersonID, PersonAge,ArriveHTime, Mode FROM TravelBehaviour
WHERE Mode = 1
INTO TeVoet;

SELECT HouseholdID, PersonID, PersonAge,ArriveHTime, Mode FROM TravelBehaviour
WHERE Mode = 2
INTO Fiets;

SELECT HouseholdID, PersonID, PersonAge,ArriveHTime, Mode FROM TravelBehaviour
WHERE Mode = 5
INTO Auto;

-- Een window aanmaken
CREATE WINDOW WindowMode
(SIZE 20 ADVANCE 1 TUPLES);

-- Schema creëren voor outputstromen
CREATE SCHEMA stroom(GemiddeldeAankomsttijd DOUBLE);

-- Outputstromen aanmaken per transportwijze
CREATE OUTPUT STREAM KindTeVoet stroom;
```

```
CREATE OUTPUT STREAM KindMetDeFiets stroom;
CREATE OUTPUT STREAM KindMetDeAuto stroom;

-- Events toevoegen aan outputstromen met de gemiddelde vertrektijd
SELECT avg(ArriveHTime)AS GemiddeldeAankomsttijd FROM TeVoet[WindowMode]
INTO KindTeVoet;

SELECT avg(ArriveHTime) AS GemiddeldeAankomsttijd FROM Fiets[WindowMode]
INTO KindMetDeFiets;

SELECT avg(ArriveHTime)AS GemiddeldeAankomsttijd FROM Auto[WindowMode]
WHERE Mode = 5
INTO KindMetDeAuto;
```

Dit project in StreamBase doet hetzelfde als het project in Coral8, alleen nu voor de aankomsttijden ’s avonds. Om hetzelfde effect te krijgen als bij Coral8, moet er achter ‘ADVANCE’ 1 staan zodat er telkens wanneer er een nieuw event in het scherm verschijnt een oud event verdwijnt. Indien hier 20 had gestaan, zouden de events in groepen van 20 in het scherm komen en uit het scherm verdwijnen. Er wordt telkens weer de gemiddelde aankomsttijd berekend op 20 events per transportwijze. Het nadeel bij StreamBase is dat nu de drie outputstromen niet naast elkaar getoond kunnen worden. In bijlage 21 zijn 3 screenshots gemaakt per transportwijze. Hier is ook duidelijk weer zichtbaar dat de gemiddelde aankomsttijd met de auto later is dan met de fiets en te voet.

HOOFDSTUK 9. Conclusies

In dit laatste hoofdstuk worden er een aantal conclusies geformuleerd. Eerst volgen een aantal theoretische conclusies. Daarna wordt er een conclusie geformuleerd aan de hand van mijn persoonlijke ondervindingen met de gebruikte tools. Deze conclusies vormen een bondig antwoord op de onderzoeksvragen.

Events zijn niet meer weg te denken uit de bedrijfswereld. Elke stap, gebeurtenis of activiteit wordt geregistreerd in bedrijven op een bepaald tijdstip, de timestamp. Door deze events te analyseren kunnen de bedrijfsactiviteiten geoptimaliseerd worden en dat komt elk bedrijf ten goede. Het probleem is dat deze events heel snel in verschillende datastromen doorheen bedrijven stromen waardoor de analyse niet eenvoudig is. Er vinden continu nieuwe events plaats, dus de gegevens van de datastromen, waarop de queries worden uitgevoerd, veranderen voortdurend. Events worden niet altijd op dezelfde manier geregistreerd en het is ook moeilijk om patronen in deze events te vinden. Het gebeurt ook regelmatig dat er belangrijke events niet hebben plaatsgevonden. Hoe kan er nu met deze grote hoeveelheid dynamische informatie worden omgegaan? Is het mogelijk om patronen in deze events weer te geven of enkel de betekenisvolle events te selecteren?

Continue queries kunnen een oplossing bieden. Dit zijn queries die eenmalig ontwikkeld worden, maar die constant opnieuw uitgevoerd worden, telkens wanneer er nieuwe events in de datastroom verschijnen. Een continue querytaal verschilt van de gekende querytaal SQL in verschillende opzichten. SQL wordt gebruikt om betekenisvolle data uit een vaste relationele database te krijgen. Een datastroom is helemaal niet vast, maar continu. De queries moeten daarom telkens opnieuw uitgevoerd worden op nieuwe events.

Deze continue queries zijn ontstaan in het teken van CEP, complex event processing. Dit is een event-driven architectuur, speciaal ontwikkeld om met een grote hoeveelheid events per seconde om te gaan. Met behulp van CEP kunnen bedrijven real-time beslissingen nemen, wat heel cruciaal is in de real-time economie de dag van vandaag. Zo worden er sneller problemen en opportuniteiten ontdekt waardoor hier snel en effectief op gereageerd kan worden. Een standaard CEP-applicatie bestaat uit een inputbron, een verwerkingssysteem waarin de queries worden uitgevoerd en een outputadapter die de betekenisvolle events opvangt.

Er zijn daarenboven heel wat tools voorhanden voor complex event processing. Elk softwarebedrijf dat in deze sector zit, heeft wel een platform met een eigen continue querytaal ter beschikking. Dit is dan ook noodzakelijk om een goede concurrentiepositie te verwerven op

de markt. Er is meer en meer interesse in CEP-platformen die gebruik maken van continue queries. De toekomst ziet er dan ook rooskleurig uit voor deze sector. In tabel 12 worden de belangrijkste softwarebedrijven met hun querytaal weergegeven.

Tabel 12: CEP-platformen met hun querytaal

Oracle	**EPL**,CQL
StreamBase	**StreamSQL**
Coral8	**CCL**
Progress Apama	Apama EPL
Aleri	Aleri SQL

De bedoeling van deze masterproef was om een aantal platformen met hun querytalen te kiezen en deze te vergelijken. Op deze manier kan dan een antwoord geformuleerd worden op de centrale onderzoeksvraag 'welke tools er beschikbaar zijn en welke tool het efficiëntste en het meest gebruiksvriendelijk is'. De algemene werking van de platformen, de structuur van de queries en de werking in de praktijk zijn aan bod gekomen. De continue querytalen die geanalyseerd worden in deze masterproef zijn EPL, StreamSQL en CCL. Hoewel er drie querytalen beschreven zijn, zijn er maar twee platformen geanalyseerd: StreamBase (StreamSQL) en Coral8 (CCL). De reden waarom er geen platform voor EPL gebruikt wordt, is simpelweg omdat er geen platform voor EPL beschikbaar was. Het platform van Progress Apama is ook niet zomaar beschikbaar op het internet. Het platform van Aleri kan dan weer wel gemakkelijk gedownload worden. Vermits Aleri en Coral8 gefusioneerd zijn, is uiteindelijk gekozen voor het platform dat het meeste focust op continue queries: Coral8.

Eerst en vooral wordt de algemene werking van de twee platformen bestudeerd. Op het eerste zicht zijn er heel wat gelijkenissen tussen de twee platformen. Tutorials worden onmiddellijk aangeboden door beide platformen om beginners wegwijs te maken in het platform. Er wordt bij beide platformen gebruik gemaakt van verschillende schermen met elk hun eigenschappen (vb. flow view, edit view, document editor,…) zodat de gebruiker gemakkelijk kan navigeren in het platform. Het grote verschil is wel dat bij Coral8 alle schermen aan elkaar gelinkt zijn en één geheel vormen waarin alles automatisch wordt geüpdatet en gesaved. Bij StreamBase wordt er gebruik gemaakt van tabbladen en zijn de schermen niet automatisch met elkaar verbonden. Er wordt hier ook meer de nadruk gelegd op het werken met de event flow editor in plaats van met de StreamSQL editor. Het platform van Coral8 is dus iets efficiënter en gemakkelijker met een sterkere focus op de continue querytaal.

Uit de vergelijkende studie van de querytalen zelf blijkt vervolgens dat de verschillende continue querytalen erg op elkaar lijken en dat ze allemaal gelinkt zijn aan SQL. De standaard queries van SQL worden door elke continue querytaal bijna letterlijk overgenomen: 'SELECT', 'FROM',

'WHERE', 'HAVING', 'GROUP BY', 'ORDER BY',…. Elke bewerking of opdracht die mogelijk is op continue datastromen, wordt wel door iedere querytaal op zijn manier geïnterpreteerd en gebruikt. In deze masterproef zijn de belangrijkste queries aan bod gekomen, maar de mogelijkheden met deze querytalen zijn nog veel uitgebreider dan dit. In dat opzicht hebben de softwarebedrijven gelijk dat ze stellen dat 'iedereen die bekend is met SQL, ook snel overweg kan met een continue querytaal'. De meeste queries zijn dan ook gemakkelijk begrijpbaar voor een doorsnee SQL-gebruiker. De structuur, conventies en redeneringen achter continue querytalen komen grotendeels overeen met deze van SQL. Er zijn natuurlijk een aantal queries die een grotere inspanning vereisen om ze onder de knie te krijgen. Er komt bij een datastroom ook meer bij kijken dan enkel de queries: de data moet op de juiste manier de applicatie binnenstromen en terug verlaten, de timestamp vraagt ook de nodige aandacht (huidige tijdstip of gespecifieerd tijdstip?), de snelheid waarmee de data in de applicatie komt,… Grote toepassingen zullen daarom zelfs voor een doorsnee SQL-gebruiker meer redeneringsvermogen en inspanning vragen. Beide tools helpen de gebruiker wel door een uitgebreid helpmenu aan te bieden. Over het algemeen zijn de twee querytalen heel duidelijk en goed uitgelegd, maar Coral8 verzorgt net iets meer uitleg door een ruimer aanbod aan voorbeelden en toepassingen aan te bieden. De voorbeelden zorgen er voor dat de theorie van de queries gemakkelijker te begrijpen is.

Na de algemene werking en de querytalen is er ten slotte een praktijkstudie uitgevoerd om een aantal belangrijke toepassingen van continue querytalen te testen. Het splitsen van stromen, het zoeken naar patronen, het niet plaatsvinden van events en sliding windows zijn aan bod gekomen. De dataset waarmee gewerkt wordt, het verplaatsingsgedrag van gezinsleden, is eigenlijk geen referentie voor datastromen die normaal gebruikt worden in deze platformen. Het is wel een goede dataset geweest om kennis te maken met continue querytalen en feeling te krijgen met de platformen. Het ingeven van de queries gebeurt bij beide platformen efficiënt met een ondersteuning van autocompletion. Deze autocompletion is wel veel uitgebreider bij StreamBase dan bij Coral8. Ook het weergeven van fouten gebeurt bij StreamBase beter, maar globaal gezien zijn beide tools goed hanteerbaar en efficiënt om mee te werken. Het uitwerken van de projecten gebeurde, afhankelijk van de toepassing, de ene keer vlotter bij Coral8 en de andere keer bij StreamBase. Het zoeken naar patronen en het niet plaatsvinden van events gaat gemakkelijker bij Coral8, terwijl de toepassing van sliding windows efficiënter verliep bij StreamBase.

Het uiteindelijke doel van de platformen is om een grote hoeveelheid events te verwerken en enkel de betekenisvolle events uit de applicatie te krijgen met behulp van de queries. Op dat vlak wordt er met beide tools hetzelfde resultaat bekomen. De resultaten van de queries kunnen opgevangen worden door een outputadapter zodat deze gebruikt kunnen worden voor verdere analyse. Indien naar de interne werking van de applicatie wordt gekeken, is er meer en

duidelijkere informatie beschikbaar bij Coral8 dan bij StreamBase. Coral8 geeft statistieken weer van wat er in de applicatie gebeurt, dus over lokale stromen en windows terwijl bij StreamBase deze informatie wordt weggelaten. Om de tool zelf te analyseren, is daarom Coral8 aangenamer en efficiënter om mee te werken omdat hier gevolgd kan worden wat er juist gedaan wordt met de events die in de applicatie binnenstromen. Het is en blijft een verwerkingstool van events, dus op dat vlak zijn beide platforms goed voorzien van de juiste mogelijkheden en kenmerken.

Het is niet eenvoudig om zomaar een besluit te trekken over wat de beste, meest efficiënte tool is voor continue queries maar over heel de lijn gezien is Coral8 toch wel het meest gebruikvriendelijke en efficiënte platform. De queries worden bij Coral8 beter uitgelegd en als doorsnee gebruiker is CCL sneller aangeleerd. Dit is natuurlijk een subjectief besluit en het verschilt ook van toepassing tot toepassing zo blijkt uit de casestudie. Dit zal in de werkelijkheid niet anders zijn. Qua ondersteuning en duidelijkheid krijgt Coral8 wel zeker de voorkeur. Deze tool is sneller aangeleerd dan deze van StreamBase. Dat de focus van Coral8 meer op de querytaal ligt dan bij StreamBase heeft er zeker en vast mee te maken. Deze thesis beperkt zich tot twee platformen en drie querytalen. Om een volledig besluit te kunnen trekken, zou het natuurlijk beter zijn om ook de andere platformen op de markt te vergelijken.

BIJLAGEN

Bijlage 1

id	Naam	leeftijd	woonplaats	functie
1	Anouk	20	Jamaica	Admin
2	Philippe	30	Londen	Admin
3	Hans	18	Parijs	Moderator
4	Jacky	18	New York	Moderator
5	Jim	25	Amsterdam	Moderator
6	Danny	18	Rome	Moderator
7	Eef	27	Rio	Moderator
8	David	21	Parijs	Reporter

Tabel 13: Voorbeeld SQL

Bijlage 2: SQL datatypes

Data type (field size) (QBE)	datatype (SQL - Access)
Text	TEXT (of CHAR, CHARACTER)
Memo	LONGTEXT (of MEMO)
Number (Byte)	BYTE
Number (Integer)	SHORT (of SMALLINT)
Number (Long Integer)	LONG (of INT, INTEGER)
Number (Single)	SINGLE (of REAL)
Number (Double)	DOUBLE (of FLOAT)
Date/Time	DATETIME (of DATE, TIME, TIMESTAMP)
Currency	CURRENCY
AutoNumber (Long Integer)	COUNTER
Yes/No	BOOLEAN

Tabel 14: Datatypes SQL

Bijlage 3: SQL conventies en operatoren

A) Vergelijkingsoperatoren

=	gelijk aan
<	kleiner dan
<=	kleinder dan of gelijk aan
>	groter dan
>=	groter dan of gelijk aan
<>	verschillend van

B) Symbolen

*	Alle kolommen selecteren
‘ ‘	String
[]	Wanneer er een spatie tussen een beschrijving zit
;	Tussen elke query moet een ; gezet worden

C) Rekenkundige operatoren

+, -	Optellen en aftrekken
*, /	Vermenigvuldigen en delen

D) Logische operatoren

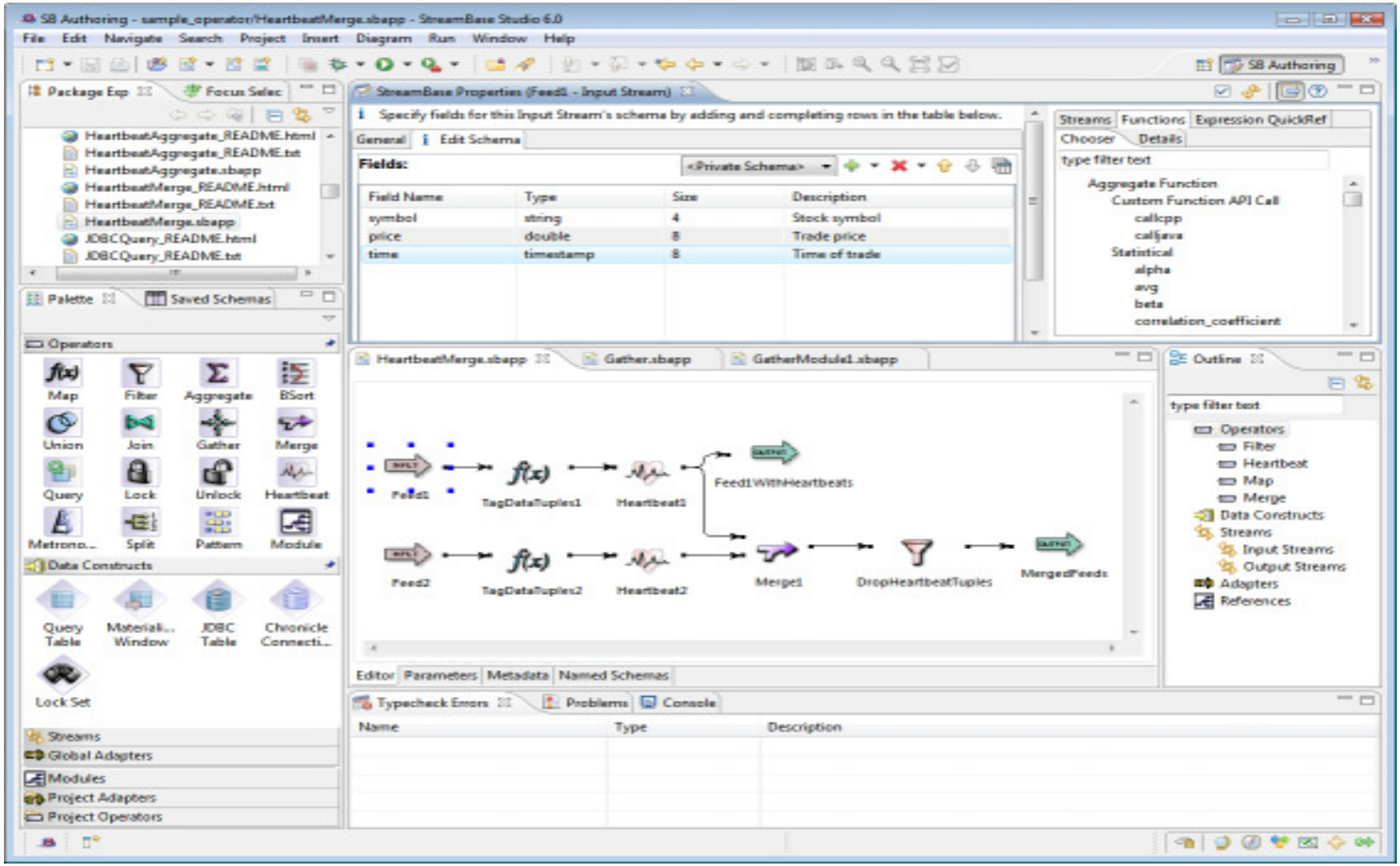
AND	Er moet aan beide voorwaarden voldaan zijn
OR	Er moet slechts aan één van beide voorwaarden voldaan zijn
NOT	Aan deze voorwaarden moet niet voldaan zijn

Bijlage 4: Gereserveerde woorden StreamSQL

advance	always	apply	asc	bsort	by	create
constant	declare	default	delete	desc	duplicate	error
every	foreach	from	gather	group	having	heartbeat
implements	import	index	input	insert	into	join
limit	lock	lockset	key	materialized	merge	metronome
offset	on	order	outer	output	parallel	parameters
partition	pattern	primary	predicate	private	public	replace
returning	schema	select	set	size	slack	stream
table	timeout	truncate	tuples	union	unlock	update
using	where	window	with	within	valid	vjoin

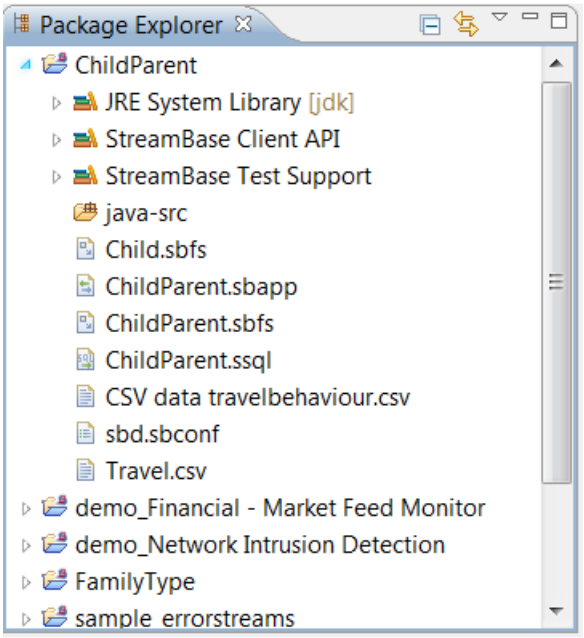
Tabel 15: Gereserveerde woorden StreamSQL

Bijlage 5: SB Author perspective van StreamBase

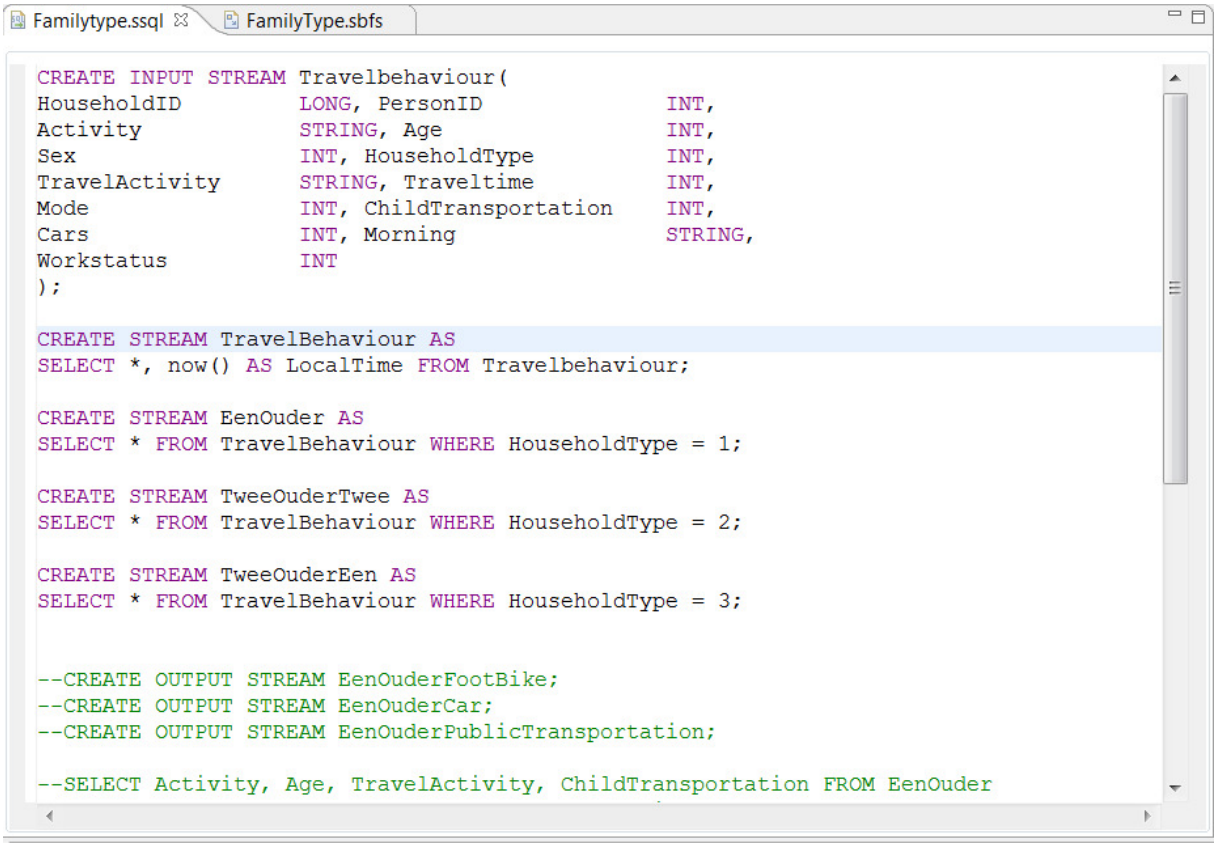


Figuur 13: SB Author perspective StreamBase

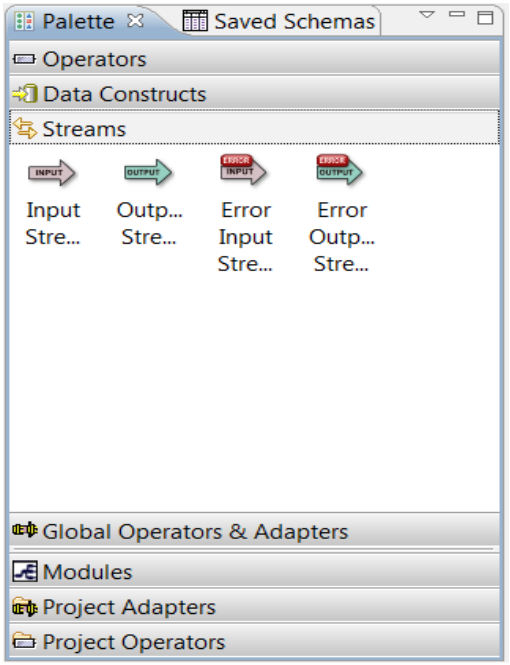
Bijlage 6: De verschillende schermen van StreamBase



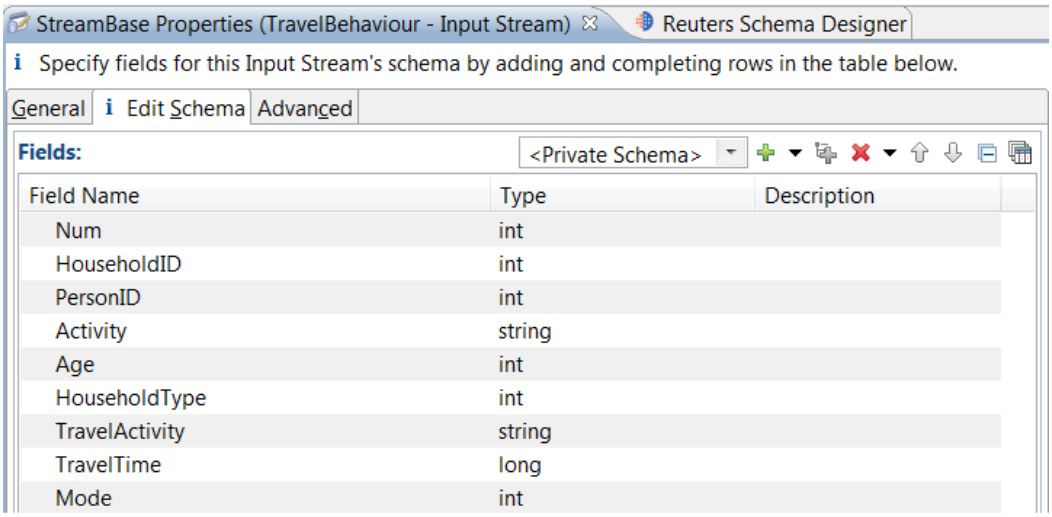
Figuur 14: Package explorer



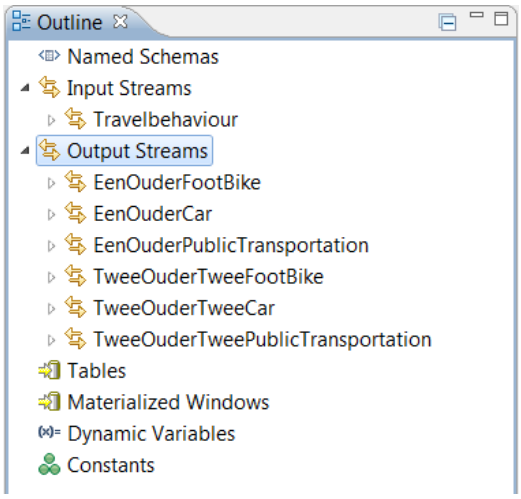
Figuur 15: Document editor



Figuur 16: Palette view

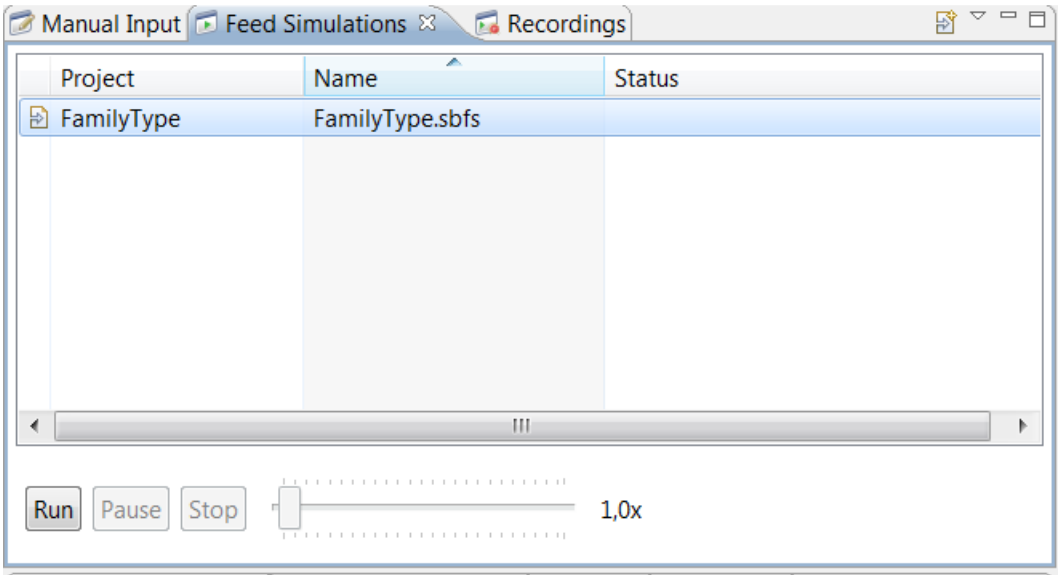


Figuur 17: Properties view

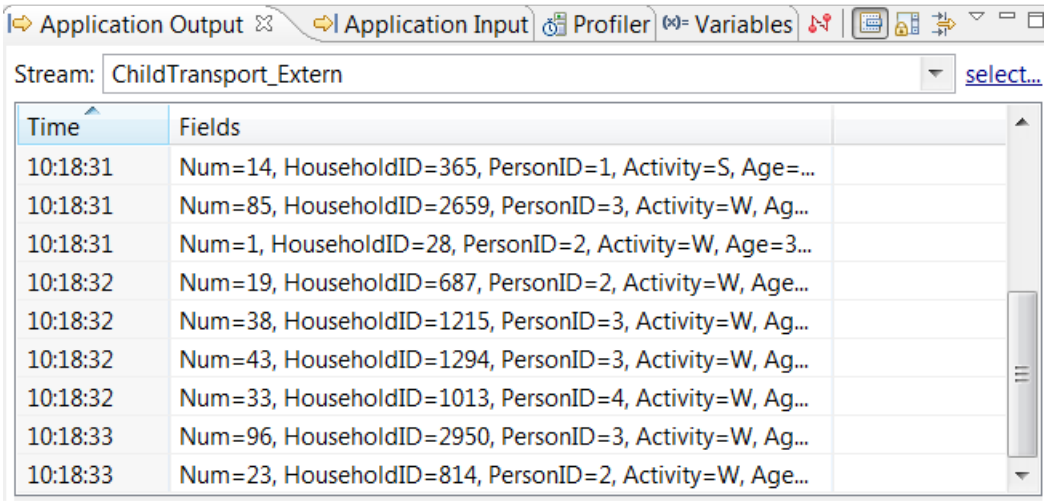


Figuur 18: Outline view

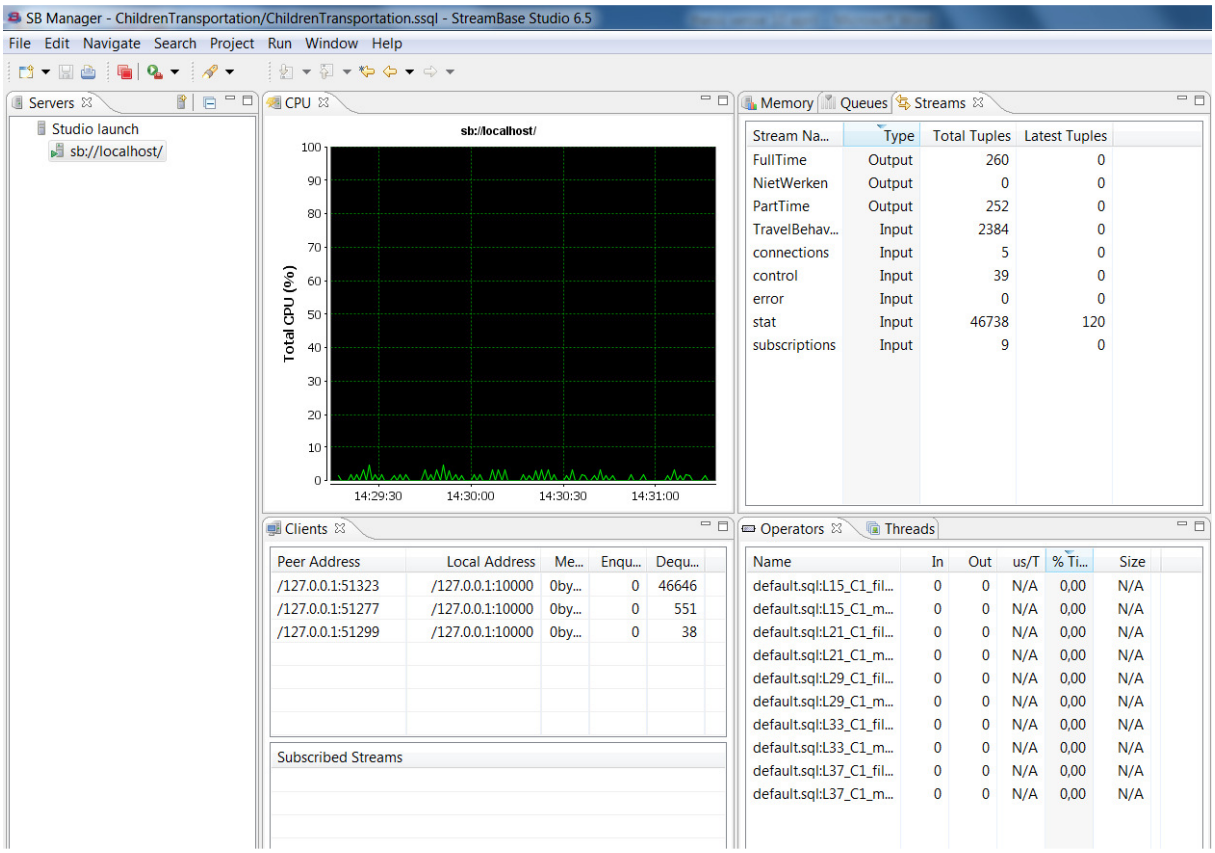
Figuur 19: Typecheck errors view



Figuur 20: Manual input/Feed simulation view



Figuur 21: Application input/output view



Figuur 22: SB Manager

Bijlage 7: Aggregatie functies StreamSQL

Tabel 16: Simpele en aggregatie functies StreamBase

Function Name and Link to Section	Category	Simple or Aggregate Function
abs	Math	Simple function
acos	Math	Simple function
aggregatelist	Aggregate to List	Aggregate function
alpha	Statistical calculations	Aggregate function
append	Lists	Simple function
asin	Math	Simple function
atan	Math	Simple function
atan2	Math	Simple function
avg (simple)	Lists	Simple function
avg (aggregate)	Statistical calculations	Aggregate function
beta	Statistical calculations	Aggregate function
black_scholes	Financial	Simple function
blob	Type conversions	Simple function
bool	Type conversions	Simple function
callcpp (simple)	External Functions (simple)	Simple function
callcpp (aggregate)	External Functions (aggregate)	Aggregate function
calljava (simple)	External Functions (simple)	Simple function
calljava (aggregate)	External Functions (aggregate)	Aggregate function
catchexception	Errors	Simple function
cbrt	Math	Simple function
ceil	Math	Simple function
closeval	Windowing	Aggregate function
coalesce	Utilities	Simple function
coalesce_tuples	Utilities	Simple function
compound_interest	Financial	Simple function
concat (simple)	Lists	Simple function
concat (aggregate)	Aggregate to List	Aggregate function
contains	Lists	Simple function
correlation_coefficient	Statistical calculations	Aggregate function
correlation_coefficientp	Statistical calculations	Aggregate function
cos	Math	Simple function
cosh	Math	Simple function

Function Name and Link to Section	Category	Simple or Aggregate Function
count	Statistical calculations	Aggregate function
count_distinct	Statistical calculations	Aggregate function
count_distinct_elements	Lists	Simple function
covariance	Statistical calculations	Aggregate function
covariancep	Statistical calculations	Aggregate function
date	Timestamps	Simple function
days	Timestamps	Simple function
dotproduct	Lists	Simple function
double	Type conversions	Simple function
emptylist	Lists	Simple function
error	Errors	Simple function
exp	Math	Simple function
exp_moving_avg	Statistical calculations	Aggregate function
expm1	Math	Simple function
filternull	Lists	Simple function
firstelement	Lists	Simple function
firstval	Windowing	Aggregate function
floor	Math	Simple function
format	Strings	Simple function
format_time	Timestamps	Simple function
from_gmtime	Timestamps	Simple function
from_localtime	Timestamps	Simple function
get_conf_param	Runtime	Simple function
getContainer	Runtime	Simple function
get_day_of_month	Timestamps	Simple function
get_day_of_week	Timestamps	Simple function
getHostName	Runtime	Simple function
get_hour	Timestamps	Simple function
getLeadershipStatus	Runtime	Simple function
get_millisecond	Timestamps	Simple function
get_minute	Timestamps	Simple function
get_month	Timestamps	Simple function
getNodeName	Runtime	Simple function
getPath	Runtime	Simple function
get_second	Timestamps	Simple function
get_year	Timestamps	Simple function

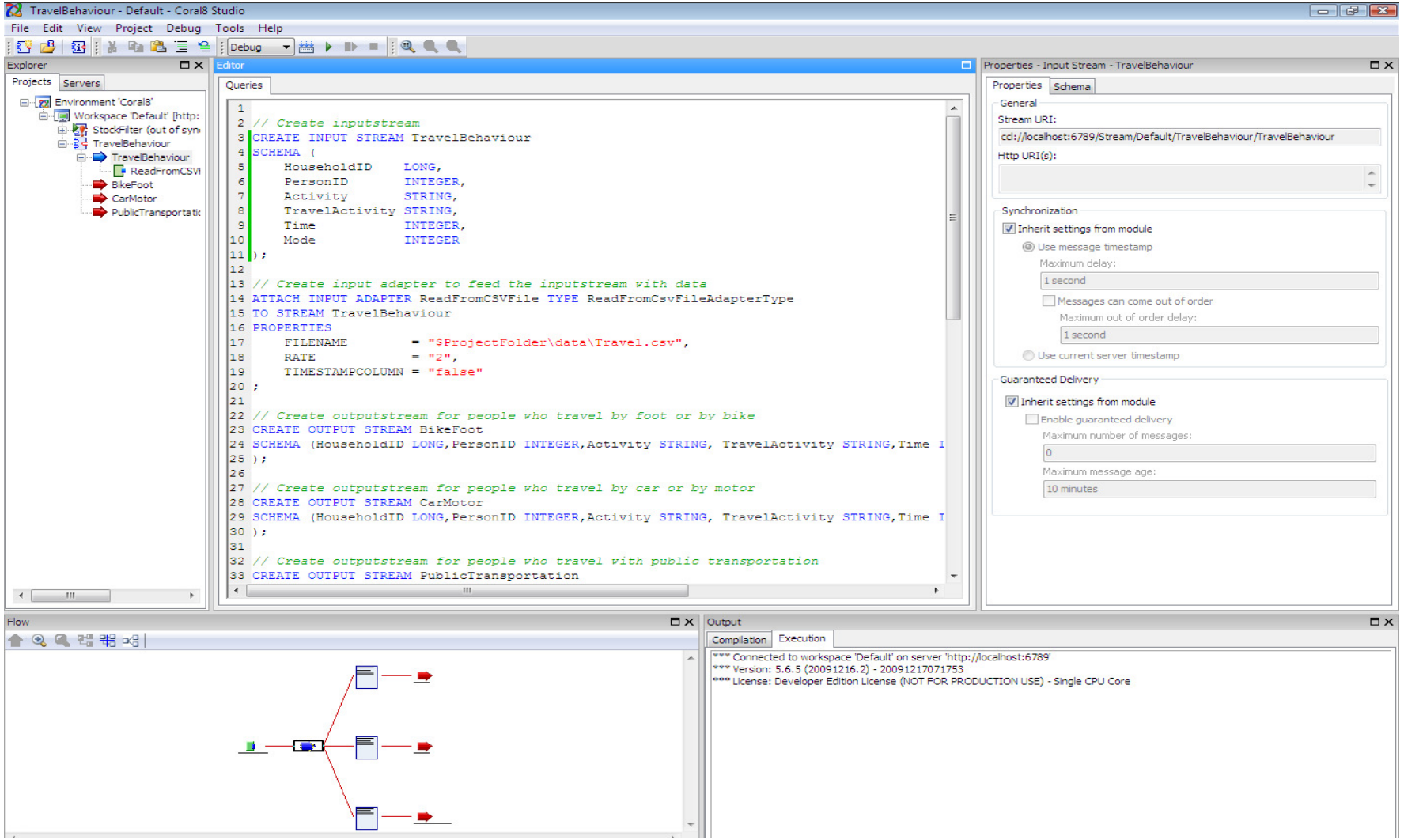
Function Name and Link to Section	Category	Simple or Aggregate Function
hash	Type conversions	Simple function
hours	Timestamps	Simple function
indexof	Utilities	Simple function
inet_aton	Internet	Simple function
inet_ntoa	Internet	Simple function
insertelement	Lists	Simple function
int	Type conversions	Simple function
intercept	Statistical calculations	Aggregate function
interval	Timestamps	Simple function
isnan	NaN (not a number)	Simple function
isnull	Utilities	Simple function
lastelement	Lists	Simple function
lastindexof	Utilities	Simple function
lastval	Windowing	Aggregate function
length	Utilities	Simple function
list	Type conversions	Simple function
ln	Math	Simple function
log10	Math	Simple function
log1p	Math	Simple function
long	Type conversions	Simple function
lower	Strings	Simple function
ltrim	Strings	Simple function
max (simple)	Math	Simple function
max (aggregate)	Statistical calculations	Aggregate function
maxdouble	Math	Simple function
maxelement	Lists	Simple function
maxint	Math	Simple function
maxlong	Math	Simple function
median (simple)	Lists	Simple function
median (aggregate)	Statistical calculations	Aggregate function
milliseconds	Timestamps	Simple function
min (simple)	Math	Simple function
min (aggregate)	Statistical calculations	Aggregate function
mindouble	Math	Simple function
minelement	Lists	Simple function
minint	Math	Simple function

Function Name and Link to Section	Category	Simple or Aggregate Function
minlong	Math	Simple function
minutes	Timestamps	Simple function
nanotime	Timestamps	Simple function
new_tuple	Type conversions	Simple function
notnan	NaN (not a number)	Simple function
notnull	Utilities	Simple function
now	Timestamps	Simple function
nulllist	Lists	Simple function
openval	Windowing	Aggregate function
parse_time	Timestamps	Simple function
pow	Math	Simple function
prepend	Lists	Simple function
product (simple)	Lists	Simple function
product (aggregate)	Statistical calculations	Aggregate function
random	Math	Simple function
range	Lists	Simple function
regexmatch	Strings	Simple function
regexmatch_ignorecase	Strings	Simple function
regexsplit	Lists	Simple function
removeelement	Lists	Simple function
replaceelement	Lists	Simple function
reverse	Lists	Simple function
round	Math	Simple function
rtrim	Strings	Simple function
seconds	Timestamps	Simple function
securerandom	Math	Simple function
securitytag	Runtime	Simple function
set_day_of_month	Timestamps	Simple function
set_day_of_week	Timestamps	Simple function
set_hour	Timestamps	Simple function
set_minute	Timestamps	Simple function
set_month	Timestamps	Simple function
set_second	Timestamps	Simple function
set_year	Timestamps	Simple function
sign	Math	Simple function
sin	Math	Simple function

Function Name and Link to Section	Category	Simple or Aggregate Function
sinh	Math	Simple function
sleep	System	Simple function
slope	Statistical calculations	Aggregate function
sort	Lists	Simple function
split	Lists	Simple function
sqrt	Math	Simple function
stdev (simple)	Lists	Simple function
stdev (aggregate)	Statistical calculations	Aggregate function
stdevp (simple)	Lists	Simple function
stdevp (aggregate)	Statistical calculations	Aggregate function
strftime	Timestamps	Simple function
string	Type conversions	Simple function
strlen	Strings	Simple function
strpinterval	Timestamps	Simple function
strptime	Timestamps	Simple function
strresize	Strings	Simple function
strresizetrunc	Strings	Simple function
sublist	Lists	Simple function
substr	Strings	Simple function
sum (simple)	Lists	Simple function
sum (aggregate)	Statistical calculations	Aggregate function
systemenv	System	Simple function
systemproperty	System	Simple function
tan	Math	Simple function
tanh	Math	Simple function
throw	Errors	Simple function
time	Timestamps	Simple function
timezoneoffset	Timestamps	Simple function
timestamp	Type conversions	Simple function
today	Timestamps	Simple function
today_utc	Timestamps	Simple function
to_degrees	Math	Simple function
to_milliseconds	Timestamps	Simple function
to_radians	Math	Simple function
to_seconds	Timestamps	Simple function
trim	Strings	Simple function

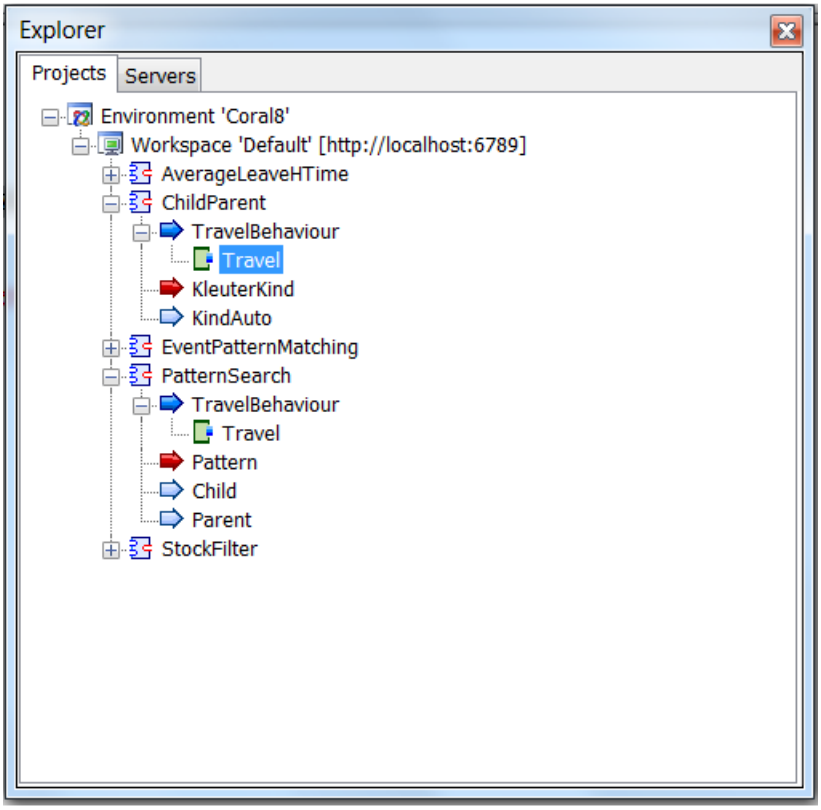
Function Name and Link to Section	Category	Simple or Aggregate Function
tuple	Type conversions	Simple function
tuple_constructor	Type conversions	Simple function
unique	Lists	Simple function
unzip	Utilities	Simple function
upper	Strings	Simple function
variance (simple)	Lists	Simple function
variance (aggregate)	Statistical calculations	Aggregate function
variancep (simple)	Lists	Simple function
variancep (aggregate)	Statistical calculations	Aggregate function
vwap	Statistical calculations	Aggregate function
weeks	Timestamps	Simple function
withmax	Statistical calculations	Aggregate function
withmin	Statistical calculations	Aggregate function
zip	Utilities	Simple function

Bijlage 8: User interface van Coral8

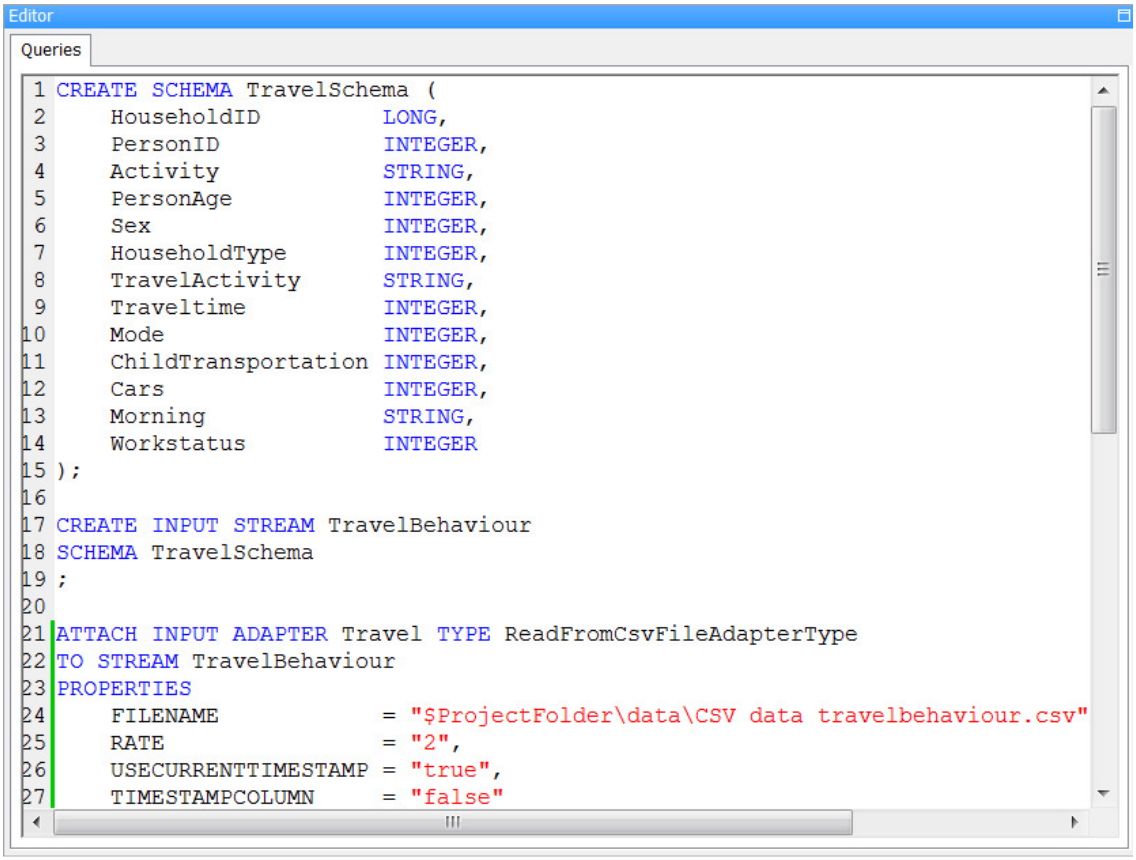


Figuur 23: User interface Coral8

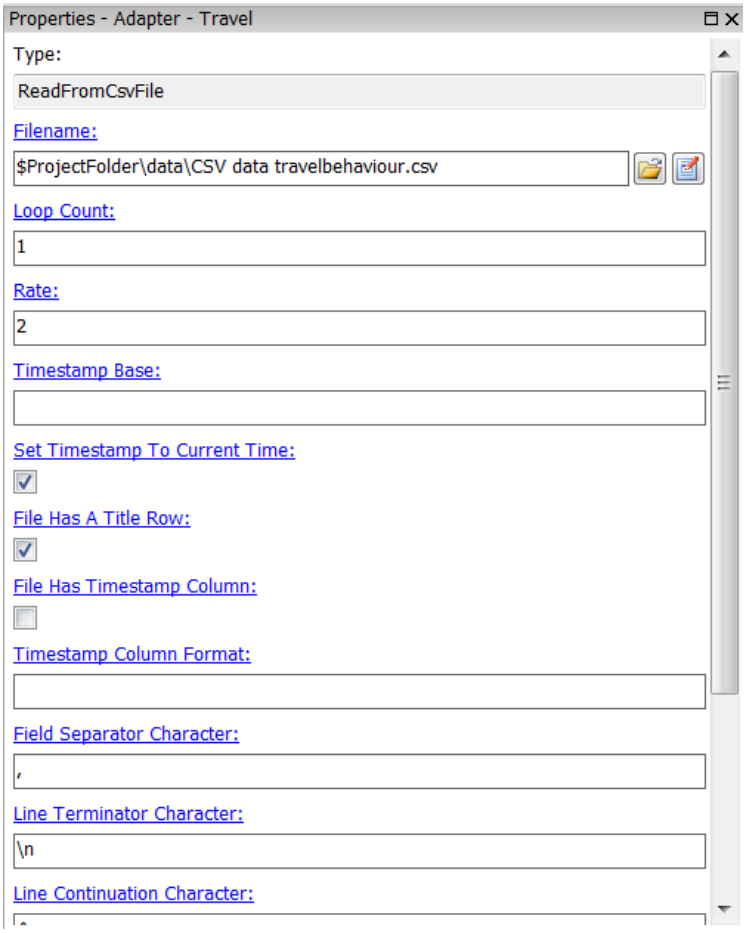
Bijlage 9: De verschillende schermen van Coral8



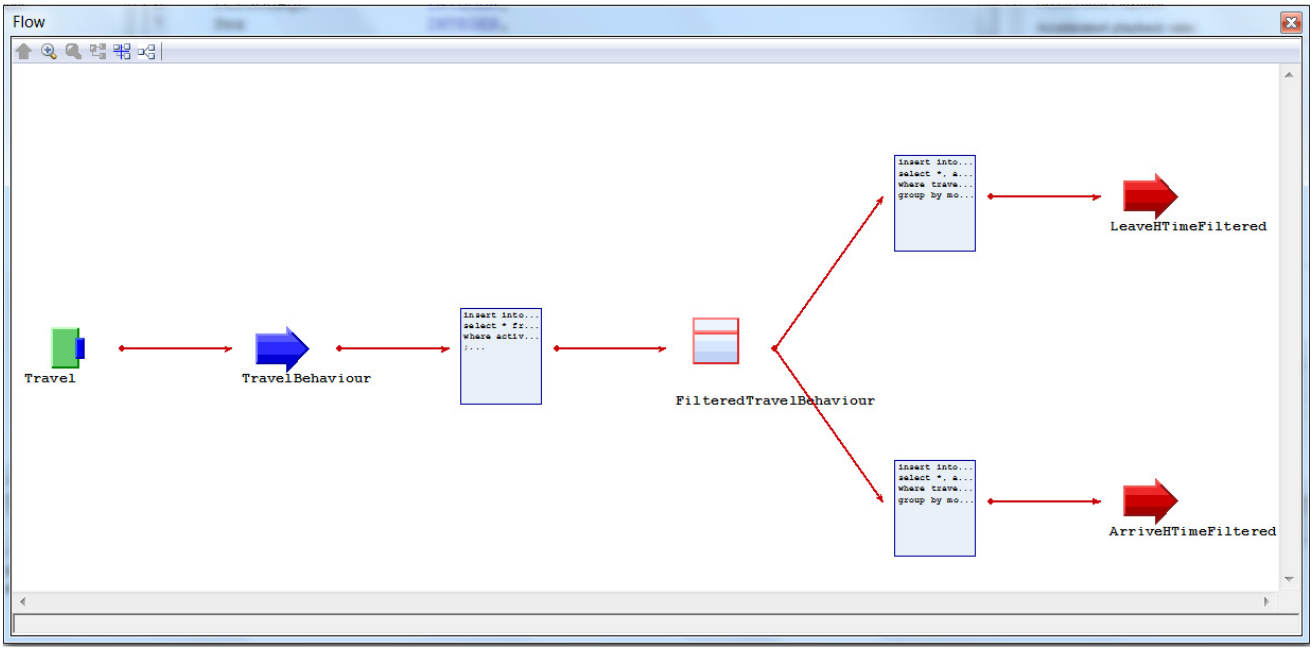
Figuur 24: Explorer view



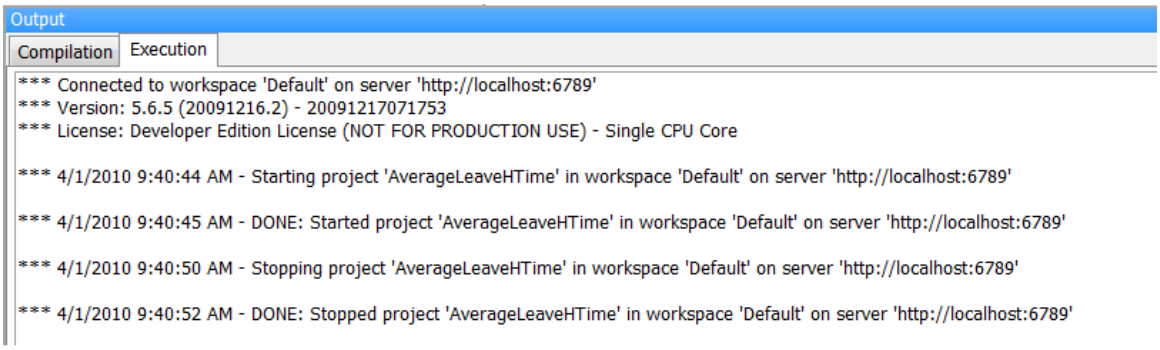
Figuur 25: Editor view



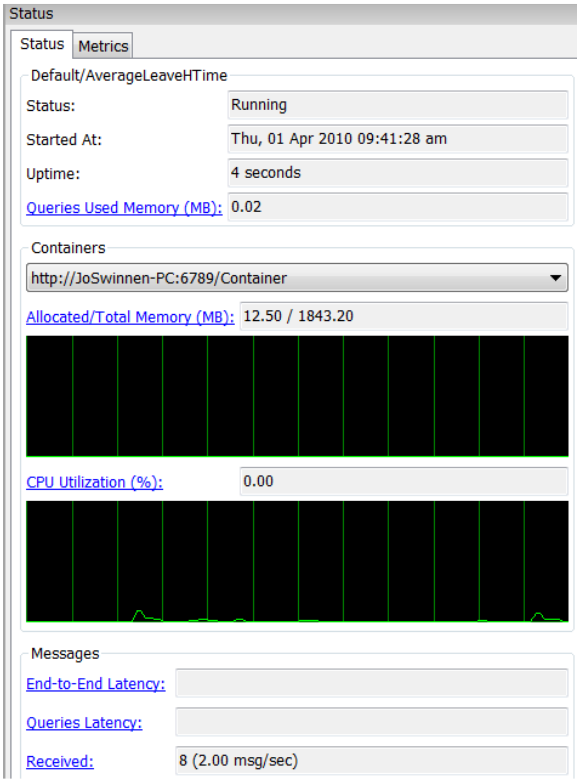
Figuur 26: Properties view



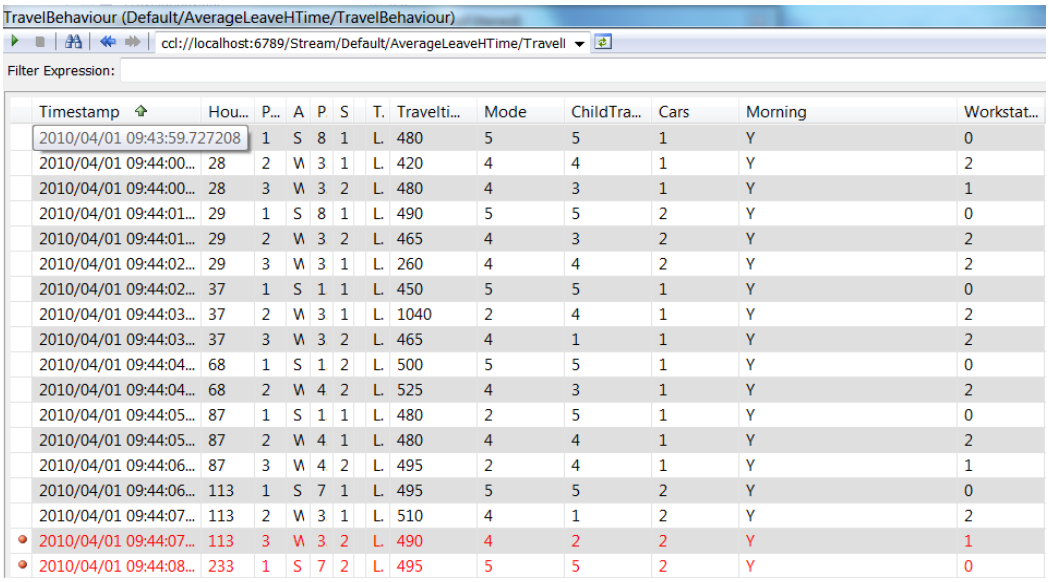
Figuur 27: Flow view



Figuur 28: Output view



Figuur 29: Status view



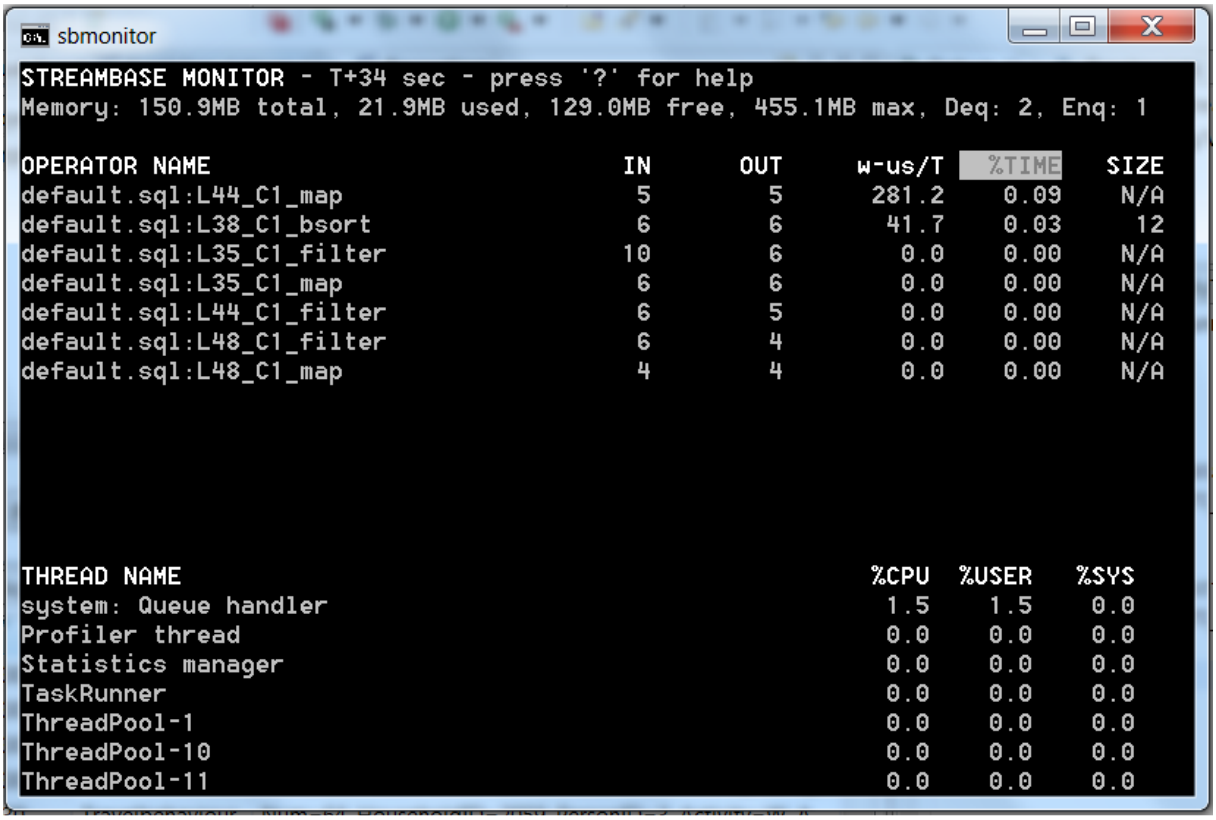
Figuur 30: Stream and window view

Bijlage 10: Gereserveerde woorden Coral8

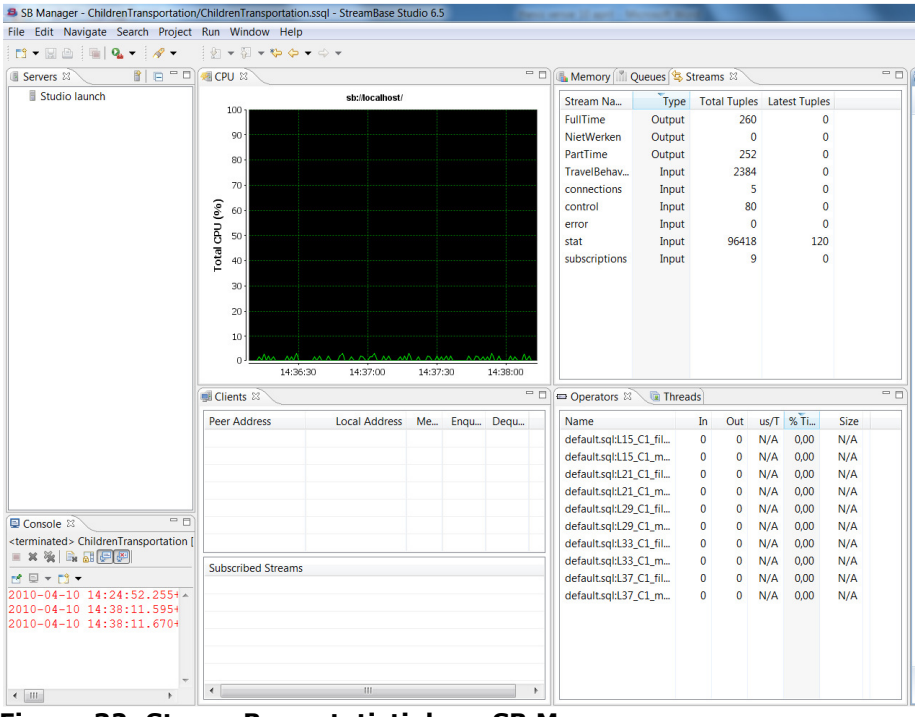
ADAPTER	AFTER	ALL	AND
ANY	AS	ASC	ASCENDING
AT	ATTACH	BEFORE	BEGIN
BREAK	BY	CASE	COLUMNS
CONTINUE	CREATE	DATABASE	DAY
DAYS	DEFAULT	DELETE	DESC
DESCENDING	DISTINCT	DO	DOWN
ELSE	ELSEIF	END	EVERY
EXECUTE	FALSE	FOR	FROM
FULL	FUNCTION	GETTIMESTAMP	GROUP
HAVING	HOUR	HOURS	IF
IMPORT	IN	INHERITS	INPUT
INSERT	INTO	IS	JOIN
KEEP	LARGEST	LAST	LEFT
LIKE	LIMIT	LOAD	LOCAL
MATCHING	MICROSECOND	MILLISECOND	MINUTe
MIRROR	MOD	MODULE	NO
NOT	NULL	OFFSET	ON
OR	ORDER	OTHERWISE	OUTER
OUTPUT	PARAMETER	PASSING	PER
PROCEDURE	PROPERTIES	QUERY	REMOTE
RETURN	RIGHT	ROW	SCHEMA
SECOND	SELECT	SET	SMALLEST
STARTING	STARTUP	STATEMENT	STREAM
THEN	TO	TRUE	UNGROUPED
UNTIL	UP	UPDATE	VARIABLE
WEEK	WHEN	WHERE	WHILE
WINDOW	WITHIN	XOR	

Tabel 17: Gereserveerde woorden Coral8

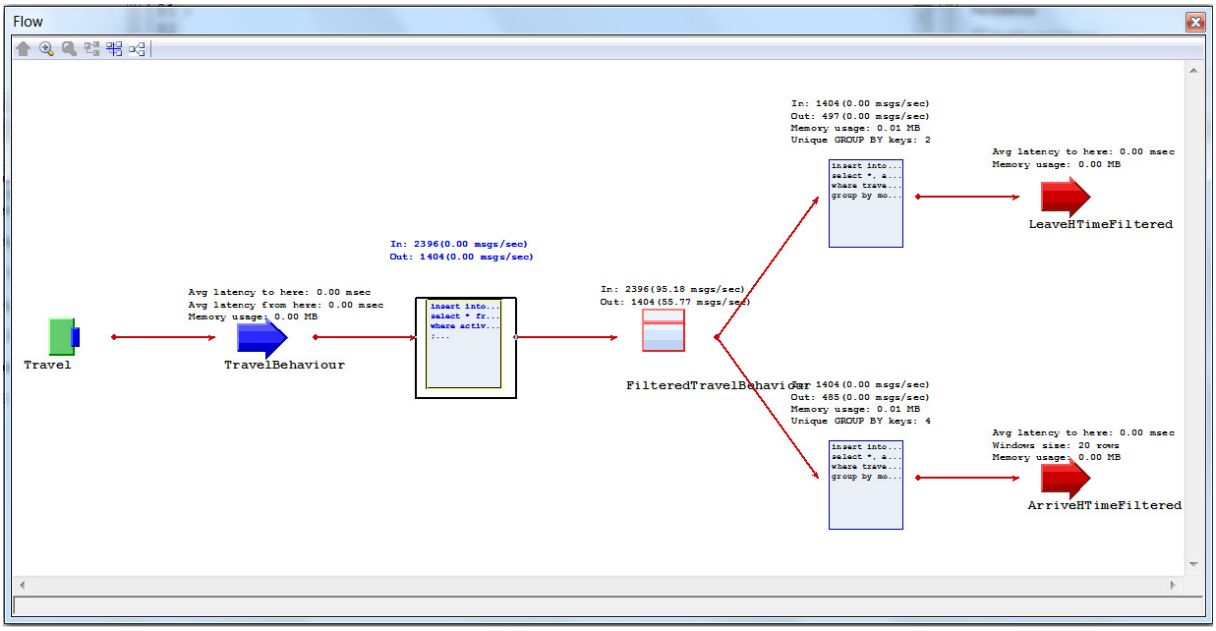
Bijlage 11: Vergelijking statistieken bij het uitvoeren van een project



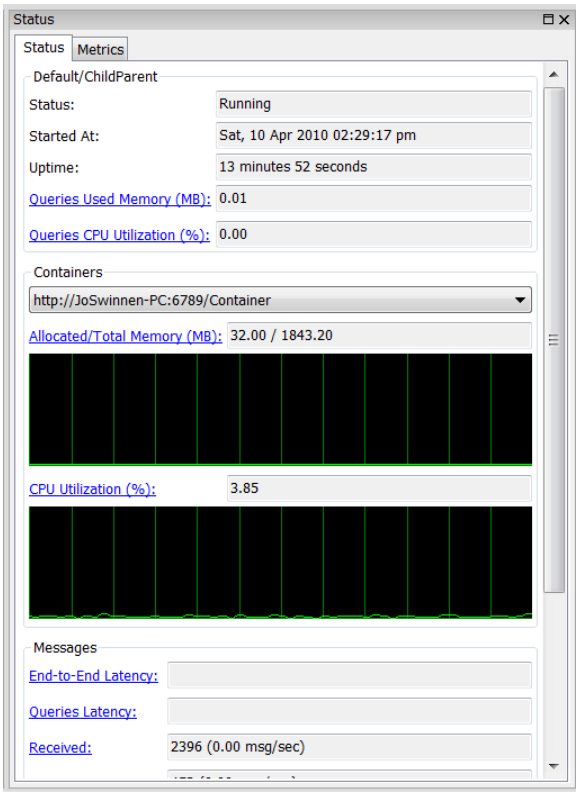
Figuur 31: StreamBase statistieken: SB Monitor



Figuur 32: StreamBase statistieken: SB Manager

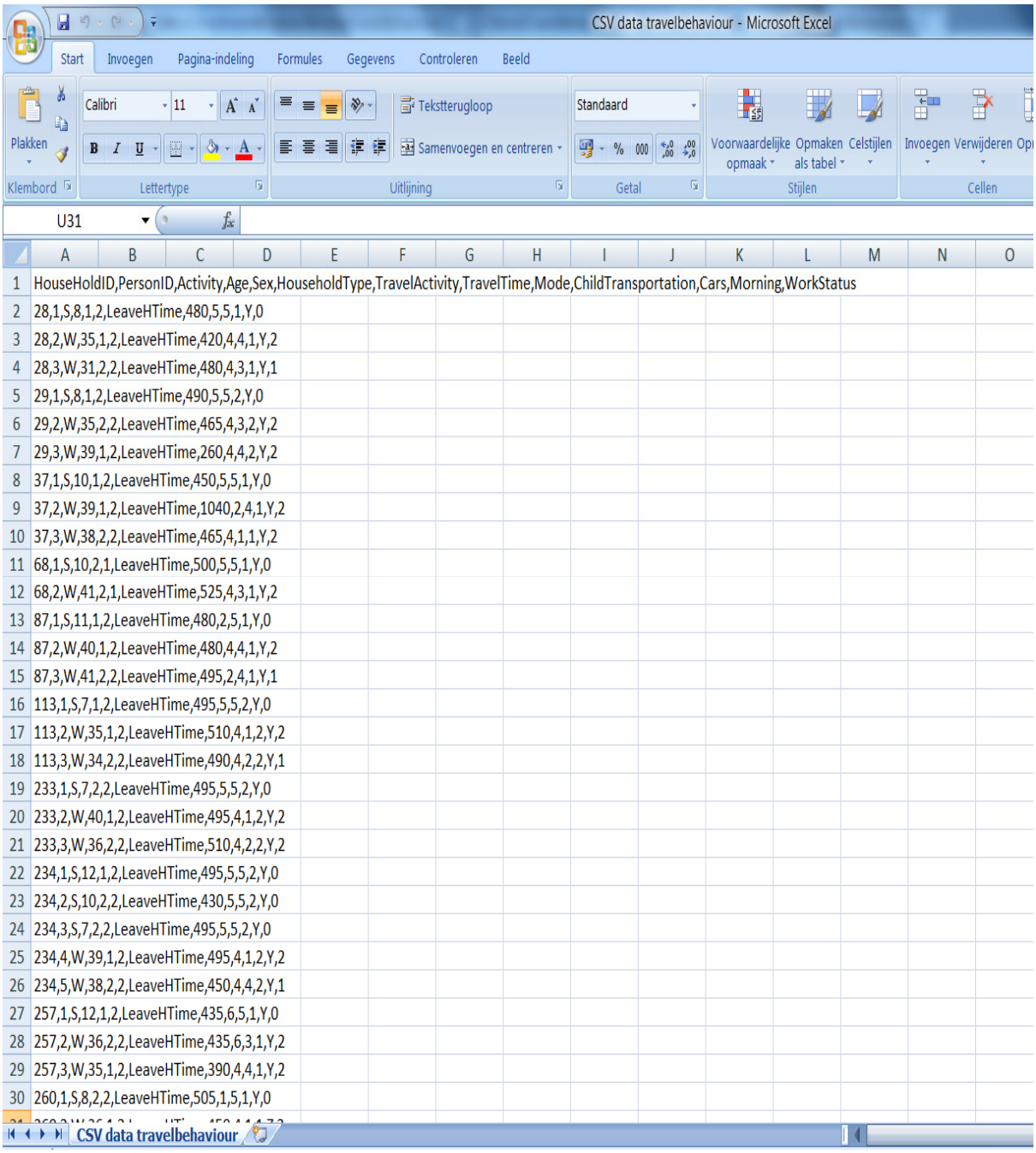


Figuur 33: Coral8 statistieken flow view



Figuur 34: Coral8 statistieken status view

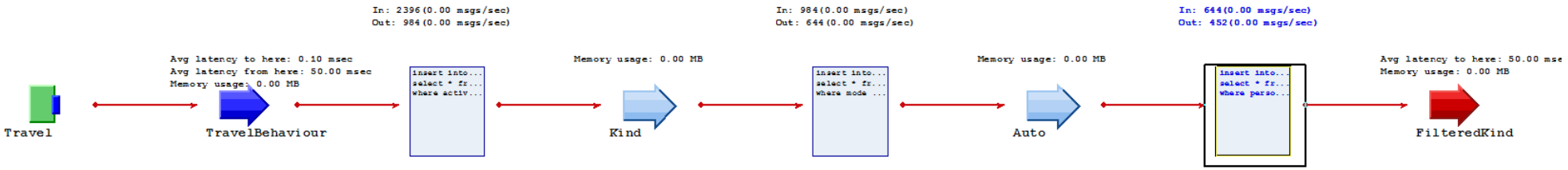
Bijlage 12: CSV-bestand van de datastroom 'verplaatsingsgedrag'



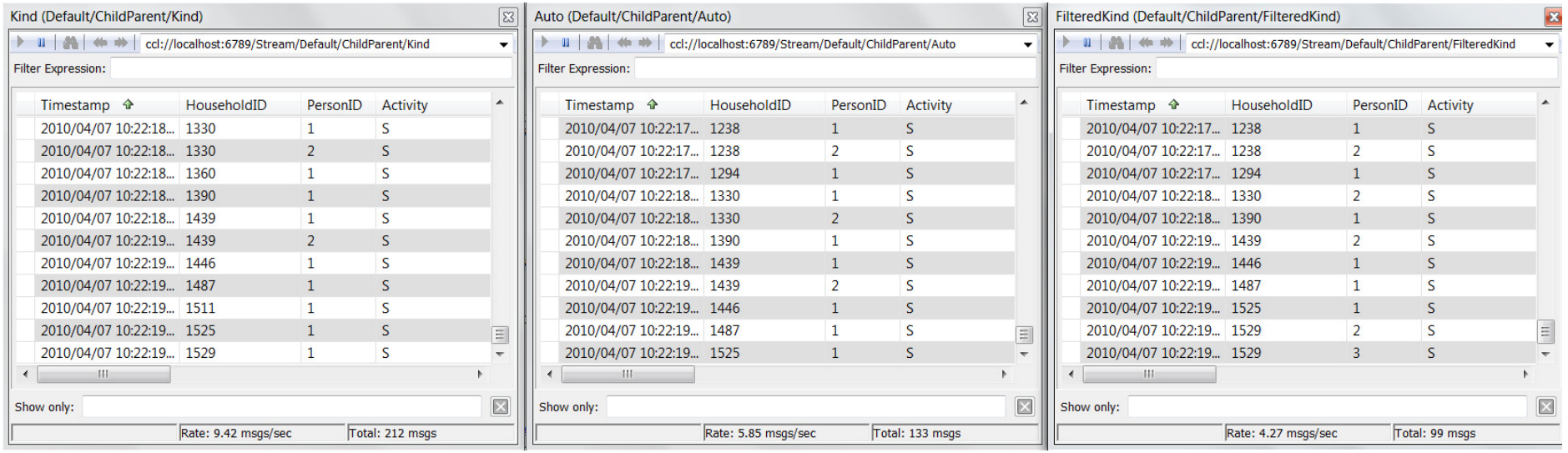
	A
1	HouseHoldID,PersonID,Activity,Age,Sex,HouseholdType,TravelActivity,TravelTime,Mode,ChildTransportation,Cars,Morning,WorkStatus
2	28,1,S,8,1,2,LeaveHTime,480,5,5,1,Y,0
3	28,2,W,35,1,2,LeaveHTime,420,4,4,1,Y,2
4	28,3,W,31,2,2,LeaveHTime,480,4,3,1,Y,1
5	29,1,S,8,1,2,LeaveHTime,490,5,5,2,Y,0
6	29,2,W,35,2,2,LeaveHTime,465,4,3,2,Y,2
7	29,3,W,39,1,2,LeaveHTime,260,4,4,2,Y,2
8	37,1,S,10,1,2,LeaveHTime,450,5,5,1,Y,0
9	37,2,W,39,1,2,LeaveHTime,1040,2,4,1,Y,2
10	37,3,W,38,2,2,LeaveHTime,465,4,1,1,Y,2
11	68,1,S,10,2,1,LeaveHTime,500,5,5,1,Y,0
12	68,2,W,41,2,1,LeaveHTime,525,4,3,1,Y,2
13	87,1,S,11,1,2,LeaveHTime,480,2,5,1,Y,0
14	87,2,W,40,1,2,LeaveHTime,480,4,4,1,Y,2
15	87,3,W,41,2,2,LeaveHTime,495,2,4,1,Y,1
16	113,1,S,7,1,2,LeaveHTime,495,5,5,2,Y,0
17	113,2,W,35,1,2,LeaveHTime,510,4,1,2,Y,2
18	113,3,W,34,2,2,LeaveHTime,490,4,2,2,Y,1
19	233,1,S,7,2,2,LeaveHTime,495,5,5,2,Y,0
20	233,2,W,40,1,2,LeaveHTime,495,4,1,2,Y,2
21	233,3,W,36,2,2,LeaveHTime,510,4,2,2,Y,2
22	234,1,S,12,1,2,LeaveHTime,495,5,5,2,Y,0
23	234,2,S,10,2,2,LeaveHTime,430,5,5,2,Y,0
24	234,3,S,7,2,2,LeaveHTime,495,5,5,2,Y,0
25	234,4,W,39,1,2,LeaveHTime,495,4,1,2,Y,2
26	234,5,W,38,2,2,LeaveHTime,450,4,4,2,Y,1
27	257,1,S,12,1,2,LeaveHTime,435,6,5,1,Y,0
28	257,2,W,36,2,2,LeaveHTime,435,6,3,1,Y,2
29	257,3,W,35,1,2,LeaveHTime,390,4,4,1,Y,2
30	260,1,S,8,2,2,LeaveHTime,505,1,5,1,Y,0

Figuur 35: CSV-bestand verplaatsingsgedrag

Bijlage 13: Project 1, resultaten Coral8



Figuur 36: flow view project 1



Figuur 37: Stream view project 1

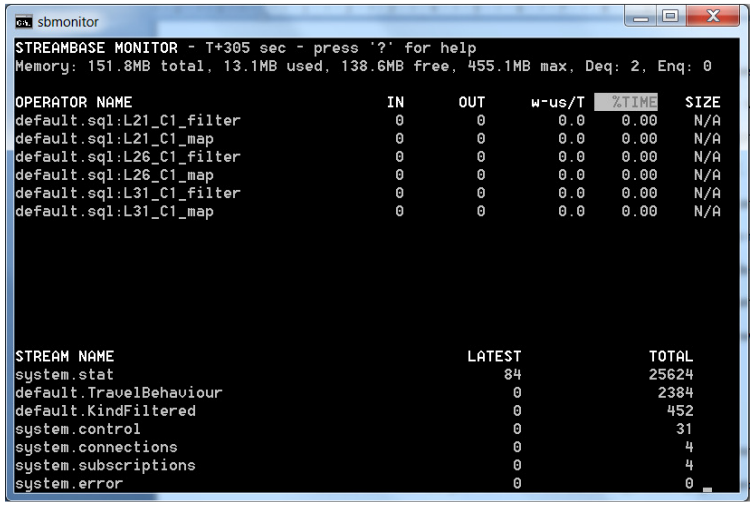
Bijlage 14: Project 1, resultaten StreamBase

Application Output | Application Input | Profiler | Variables

Stream: (All Output Streams) select...

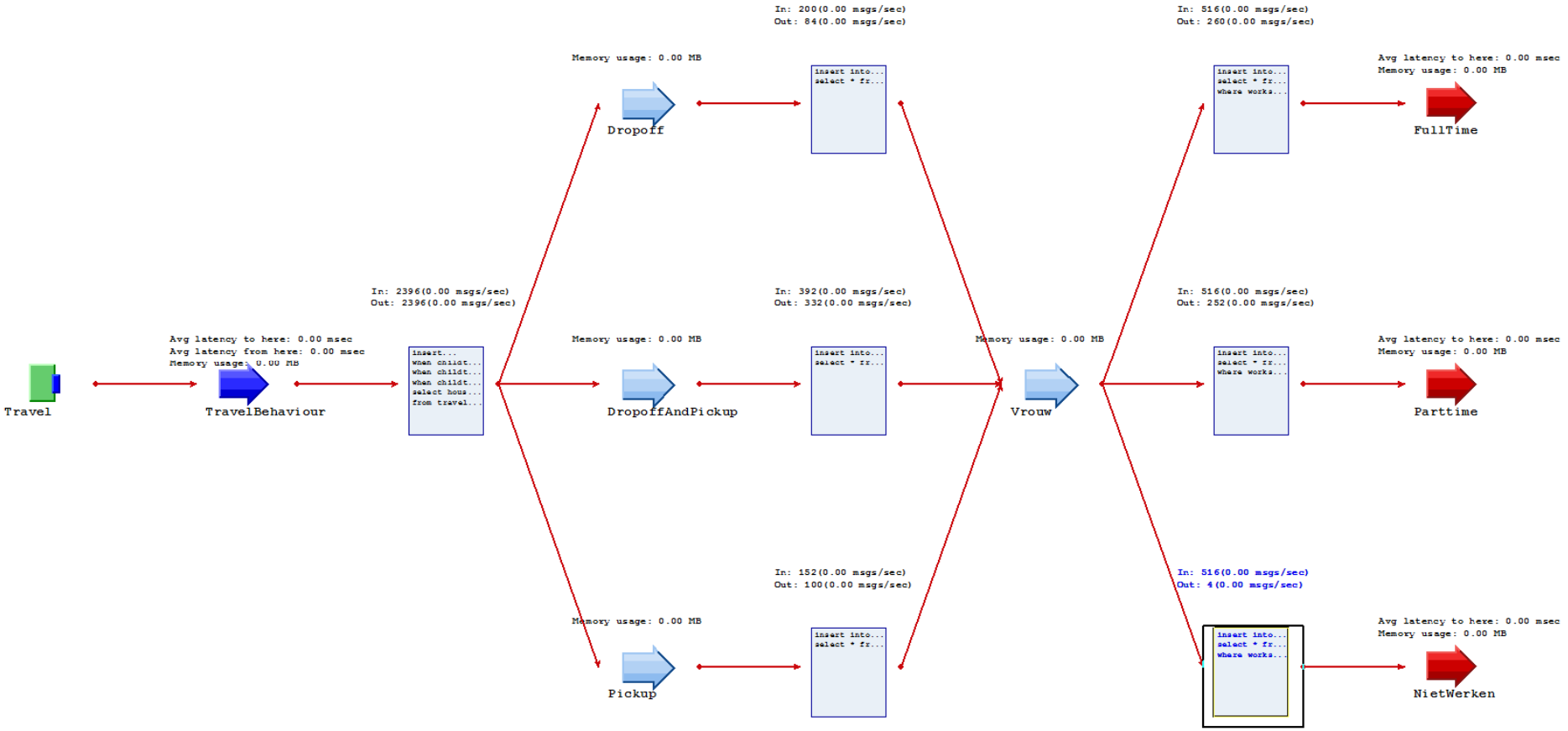
Time	Output Stream	Fields
14:20:18	KindFiltered	HouseholdID=4668, PersonID=2, Age=6, HouseholdType=2, TravelActivity=ArriveHTime, Traveltime=1040
14:20:19	KindFiltered	HouseholdID=4671, PersonID=2, Age=9, HouseholdType=2, TravelActivity=ArriveHTime, Traveltime=990
14:20:19	KindFiltered	HouseholdID=4690, PersonID=1, Age=8, HouseholdType=2, TravelActivity=ArriveHTime, Traveltime=950
14:20:19	KindFiltered	HouseholdID=4727, PersonID=1, Age=7, HouseholdType=2, TravelActivity=ArriveHTime, Traveltime=955
14:20:20	KindFiltered	HouseholdID=4765, PersonID=1, Age=7, HouseholdType=2, TravelActivity=ArriveHTime, Traveltime=945
14:20:21	KindFiltered	HouseholdID=4772, PersonID=1, Age=9, HouseholdType=2, TravelActivity=ArriveHTime, Traveltime=1040
14:20:21	KindFiltered	HouseholdID=4827, PersonID=1, Age=6, HouseholdType=2, TravelActivity=ArriveHTime, Traveltime=1060
14:20:21	KindFiltered	HouseholdID=4914, PersonID=1, Age=6, HouseholdType=2, TravelActivity=ArriveHTime, Traveltime=945
14:20:22	KindFiltered	HouseholdID=4918, PersonID=1, Age=7, HouseholdType=2, TravelActivity=ArriveHTime, Traveltime=945
14:20:22	KindFiltered	HouseholdID=4940, PersonID=1, Age=8, HouseholdType=2, TravelActivity=ArriveHTime, Traveltime=960
14:20:22	KindFiltered	HouseholdID=4944, PersonID=1, Age=8, HouseholdType=2, TravelActivity=ArriveHTime, Traveltime=970
14:20:22	KindFiltered	HouseholdID=4944, PersonID=2, Age=6, HouseholdType=2, TravelActivity=ArriveHTime, Traveltime=970
14:20:23	KindFiltered	HouseholdID=4970, PersonID=1, Age=8, HouseholdType=2, TravelActivity=ArriveHTime, Traveltime=940
14:20:24	KindFiltered	HouseholdID=30595, PersonID=1, Age=6, HouseholdType=3, TravelActivity=LeaveHTime, Traveltime=470
14:20:24	KindFiltered	HouseholdID=30595, PersonID=1, Age=6, HouseholdType=3, TravelActivity=ArriveWSTime, Traveltime=485
14:20:24	KindFiltered	HouseholdID=30595, PersonID=1, Age=6, HouseholdType=3, TravelActivity=LeaveWSTime, Traveltime=960
14:20:25	KindFiltered	HouseholdID=30595, PersonID=1, Age=6, HouseholdType=3, TravelActivity=ArriveHTime, Traveltime=970

Figuur 38: Application output project 1

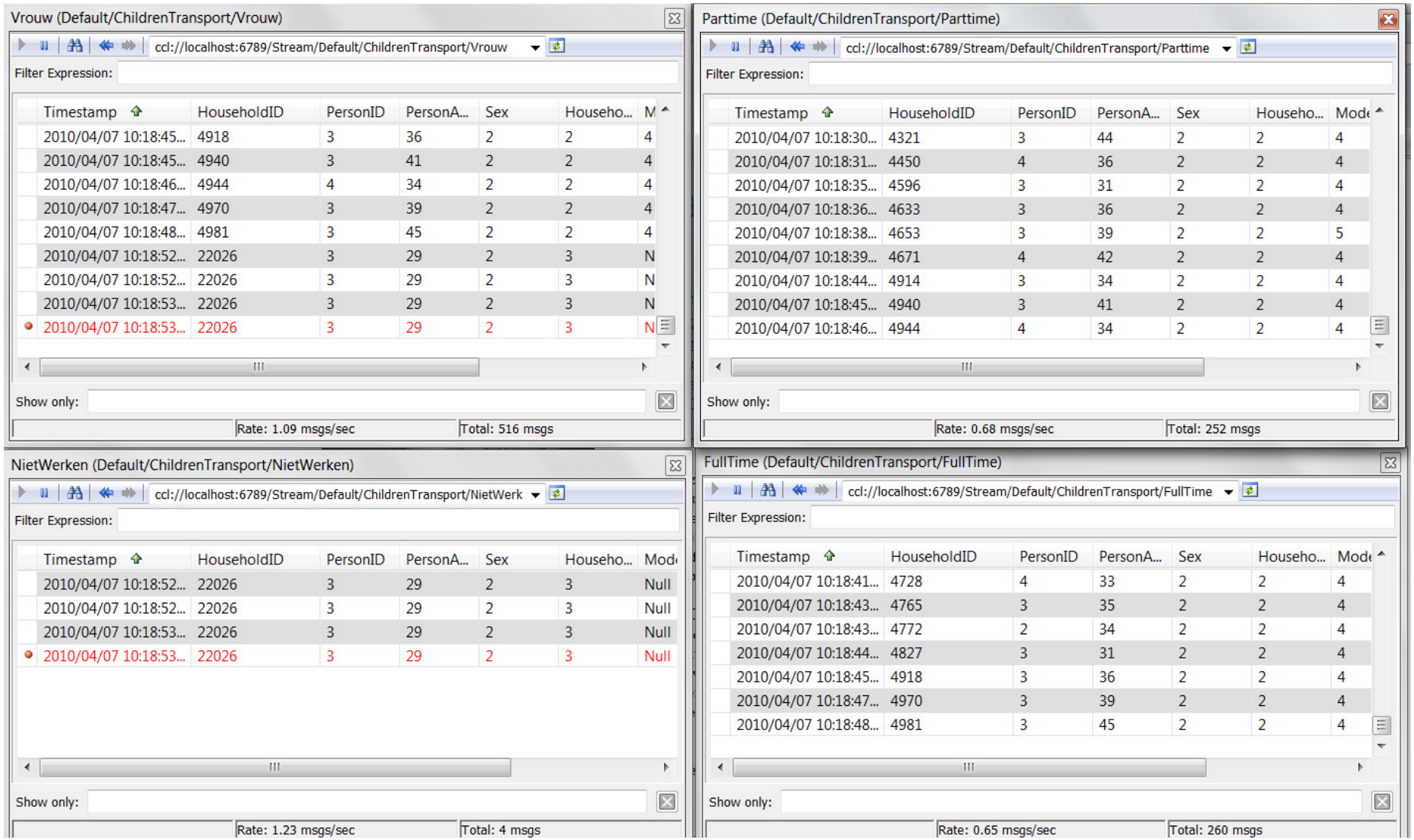


Figuur 39: sb monitor project 1

Bijlage 15: Project 2, resultaten Coral8



Figuur 40: Flow view project 2



Figuur 41: Stream view project

Bijlage 16: Project 2, resultaten StreamBase

Application Output | Application Input | Profiler | Variables

Stream: FullTime select...

Time	Fields
17:07:21	HouseholdID=3431, PersonID=4, PersonAge=42, Sex=2, HouseholdType=2, Mode=4, ChildTransportation=3, WorkStatus=2
17:07:21	HouseholdID=3433, PersonID=3, PersonAge=40, Sex=2, HouseholdType=2, Mode=4, ChildTransportation=2, WorkStatus=2
17:07:22	HouseholdID=3614, PersonID=3, PersonAge=38, Sex=2, HouseholdType=2, Mode=4, ChildTransportation=3, WorkStatus=2
17:07:23	HouseholdID=3722, PersonID=3, PersonAge=38, Sex=2, HouseholdType=2, Mode=4, ChildTransportation=3, WorkStatus=2
17:07:23	HouseholdID=3774, PersonID=2, PersonAge=39, Sex=2, HouseholdType=1, Mode=4, ChildTransportation=3, WorkStatus=2
17:07:26	HouseholdID=3924, PersonID=3, PersonAge=32, Sex=2, HouseholdType=2, Mode=4, ChildTransportation=3, WorkStatus=2
17:07:26	HouseholdID=3946, PersonID=3, PersonAge=36, Sex=2, HouseholdType=2, Mode=4, ChildTransportation=2, WorkStatus=2
17:07:30	HouseholdID=4150, PersonID=3, PersonAge=31, Sex=2, HouseholdType=2, Mode=4, ChildTransportation=3, WorkStatus=2
17:07:31	HouseholdID=4172, PersonID=4, PersonAge=34, Sex=2, HouseholdType=2, Mode=4, ChildTransportation=3, WorkStatus=2
17:07:32	HouseholdID=4207, PersonID=3, PersonAge=33, Sex=2, HouseholdType=2, Mode=4, ChildTransportation=3, WorkStatus=2
17:07:33	HouseholdID=4234, PersonID=3, PersonAge=40, Sex=2, HouseholdType=2, Mode=4, ChildTransportation=1, WorkStatus=2
17:07:35	HouseholdID=4520, PersonID=4, PersonAge=37, Sex=2, HouseholdType=2, Mode=4, ChildTransportation=2, WorkStatus=2
17:07:35	HouseholdID=4532, PersonID=3, PersonAge=46, Sex=2, HouseholdType=2, Mode=4, ChildTransportation=1, WorkStatus=2
17:07:35	HouseholdID=4534, PersonID=3, PersonAge=40, Sex=2, HouseholdType=2, Mode=4, ChildTransportation=3, WorkStatus=2
17:07:38	HouseholdID=4668, PersonID=4, PersonAge=40, Sex=2, HouseholdType=2, Mode=4, ChildTransportation=3, WorkStatus=2
17:07:38	HouseholdID=4690, PersonID=3, PersonAge=37, Sex=2, HouseholdType=2, Mode=4, ChildTransportation=1, WorkStatus=2
17:07:39	HouseholdID=4727, PersonID=3, PersonAge=29, Sex=2, HouseholdType=2, Mode=4, ChildTransportation=3, WorkStatus=2
17:07:39	HouseholdID=4728, PersonID=4, PersonAge=33, Sex=2, HouseholdType=2, Mode=4, ChildTransportation=1, WorkStatus=2

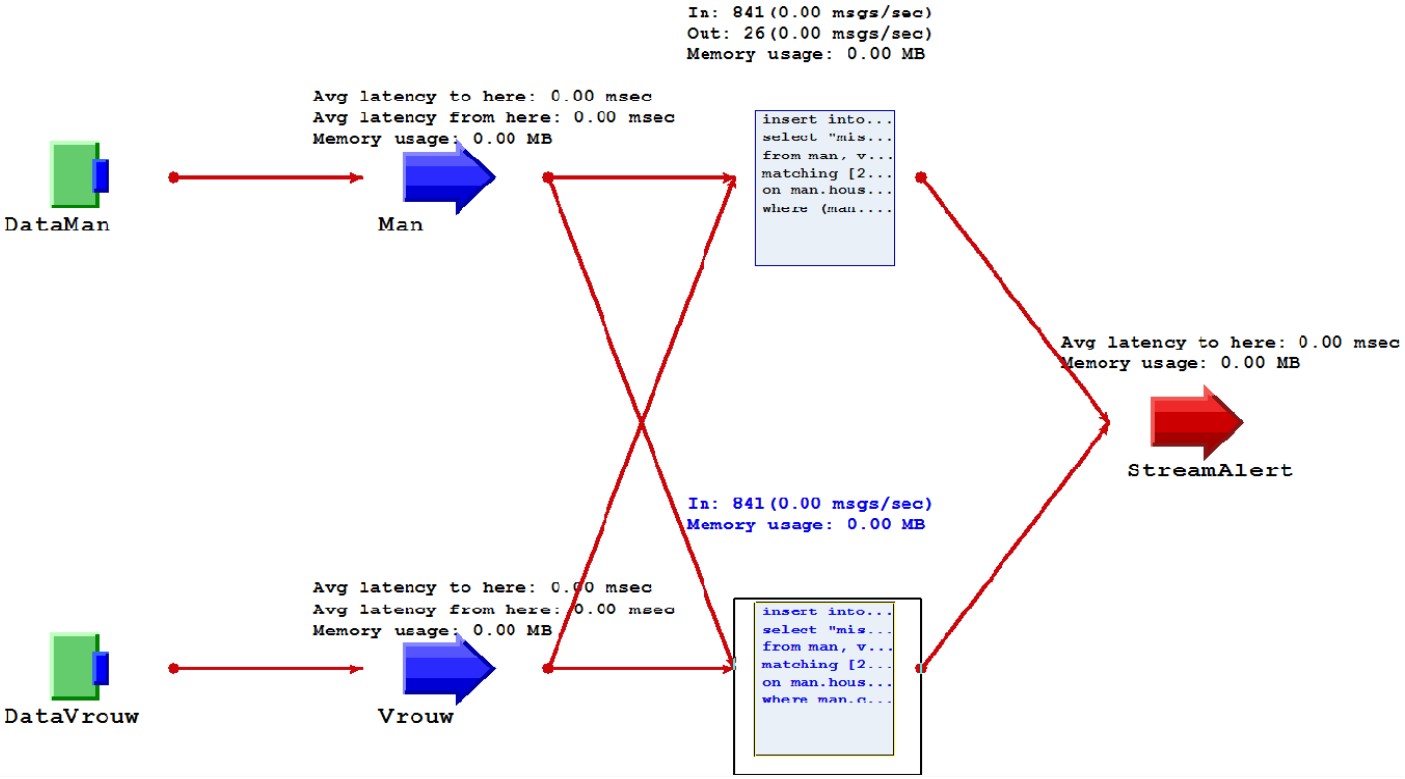
Figuur 42: Application output project 2

sbmonitor

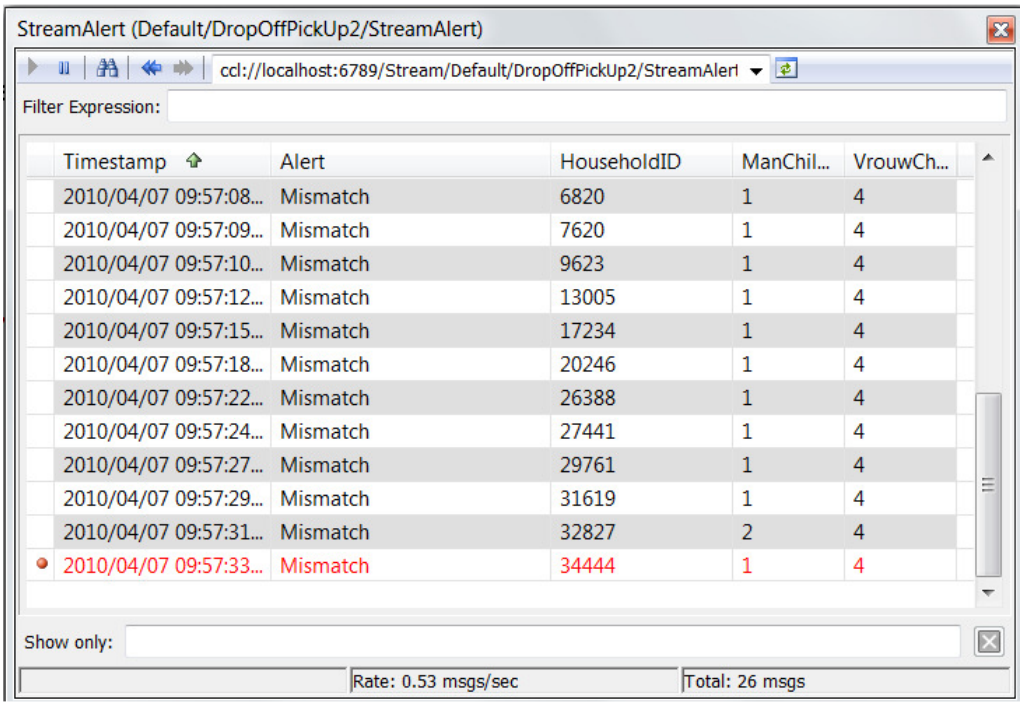
```
STREAMBASE MONITOR - T+1032 sec - press '?' for help
Memory: 154.7MB total, 37.4MB used, 117.3MB free, 455.1MB max, Deq: 2, Enq: 0

OPERATOR NAME                          IN       OUT      w-us/T   %TIME   SIZE
default.sql:L15_C1_filter               0         0         0.0    0.00    N/A
default.sql:L15_C1_map                  0         0         0.0    0.00    N/A
default.sql:L21_C1_filter               0         0         0.0    0.00    N/A
default.sql:L21_C1_map                  0         0         0.0    0.00    N/A
default.sql:L29_C1_filter               0         0         0.0    0.00    N/A
default.sql:L29_C1_map                  0         0         0.0    0.00    N/A
default.sql:L33_C1_filter               0         0         0.0    0.00    N/A
default.sql:L33_C1_map                  0         0         0.0    0.00    N/A
default.sql:L37_C1_filter               0         0         0.0    0.00    N/A
default.sql:L37_C1_map                  0         0         N/A    0.00    N/A


STREAM NAME                                    LATEST              TOTAL
system.stat                                       116             120358
default.TravelBehaviour                             0               2384
default.FullTime                                    0                260
default.PartTime                                    0                252
system.control                                      0                104
system.subscriptions                                0                  6
system.connections                                  0                  4
```

Figuur 43: sb monitor project 2

Bijlage 17: Project 3, resultaten Coral8

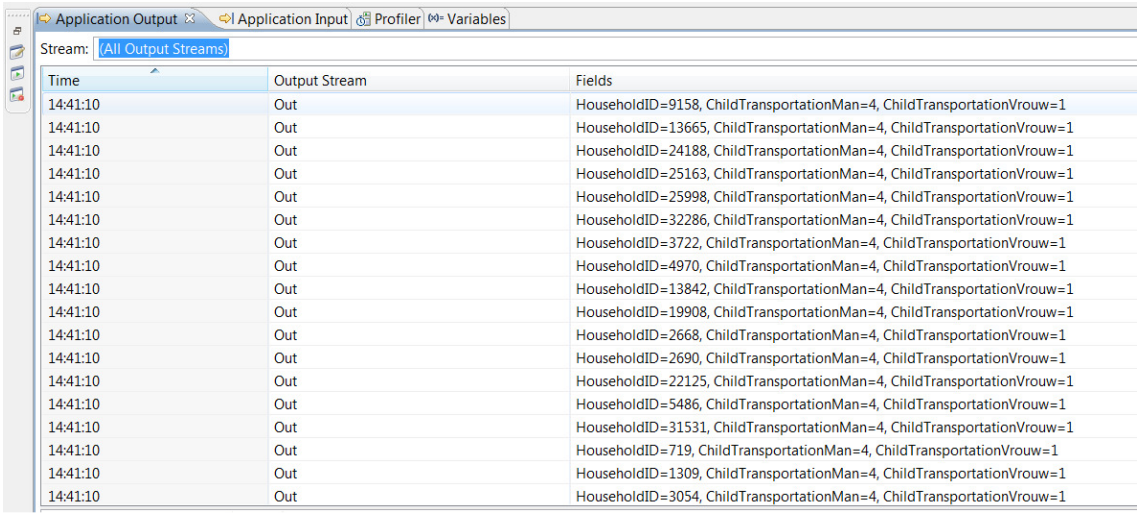


Figuur 44: Flow view project 3



Figuur 45: Stream view project 3

Bijlage 18: Casestudie 3, resultaten StreamBase



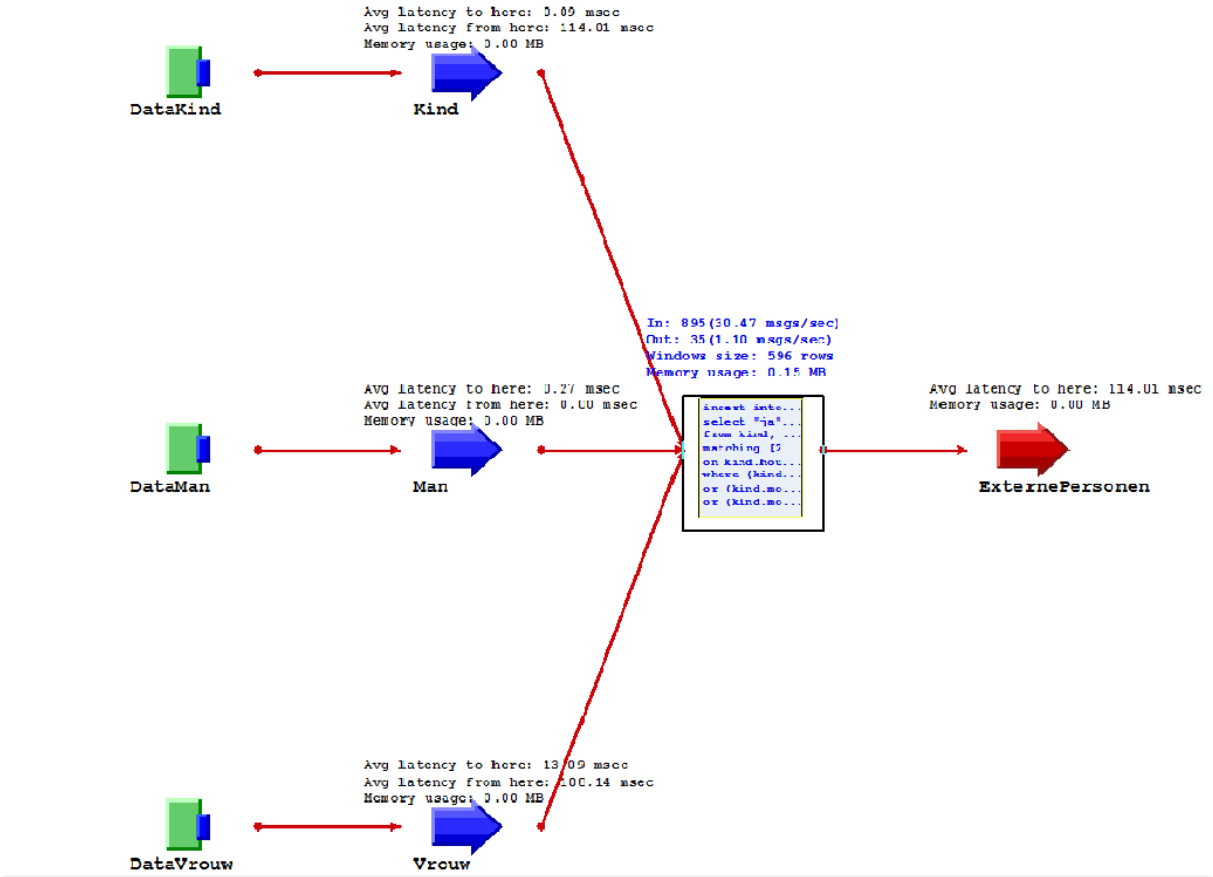
Application Output | Application Input | Profiler | Variables

Stream: (All Output Streams)

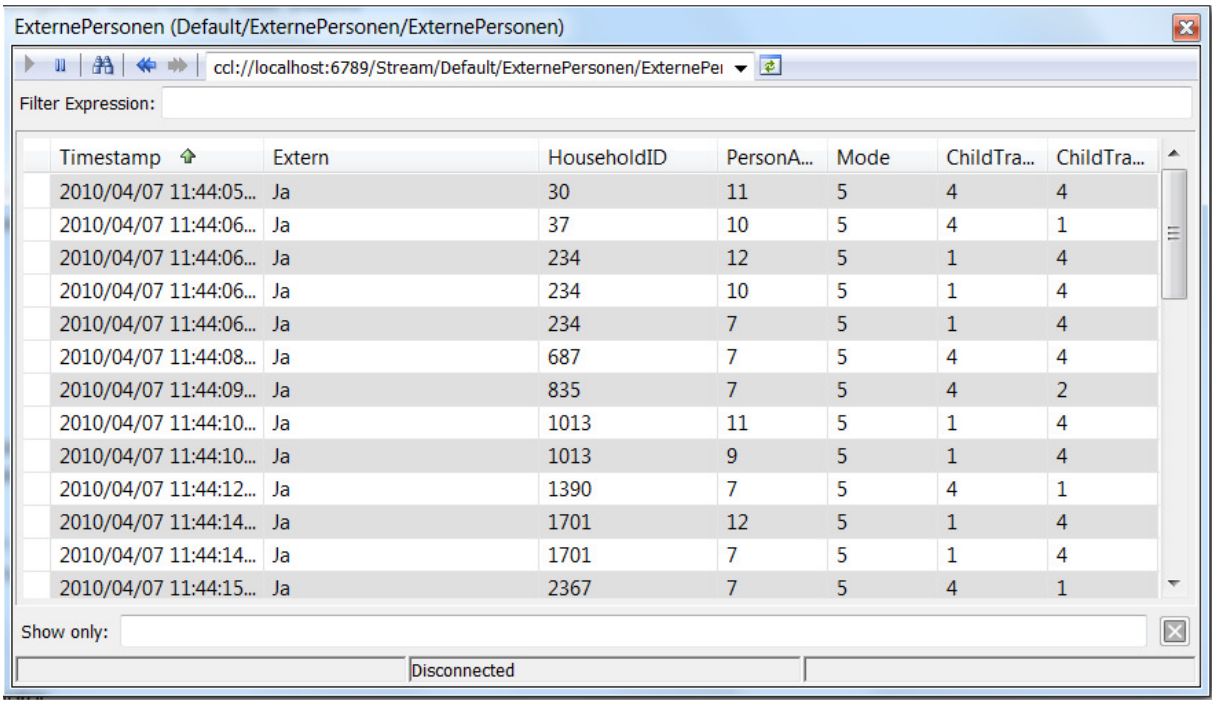
Time	Output Stream	Fields
14:41:10	Out	HouseholdID=9158, ChildTransportationMan=4, ChildTransportationVrouw=1
14:41:10	Out	HouseholdID=13665, ChildTransportationMan=4, ChildTransportationVrouw=1
14:41:10	Out	HouseholdID=24188, ChildTransportationMan=4, ChildTransportationVrouw=1
14:41:10	Out	HouseholdID=25163, ChildTransportationMan=4, ChildTransportationVrouw=1
14:41:10	Out	HouseholdID=25998, ChildTransportationMan=4, ChildTransportationVrouw=1
14:41:10	Out	HouseholdID=32286, ChildTransportationMan=4, ChildTransportationVrouw=1
14:41:10	Out	HouseholdID=3722, ChildTransportationMan=4, ChildTransportationVrouw=1
14:41:10	Out	HouseholdID=4970, ChildTransportationMan=4, ChildTransportationVrouw=1
14:41:10	Out	HouseholdID=13842, ChildTransportationMan=4, ChildTransportationVrouw=1
14:41:10	Out	HouseholdID=19908, ChildTransportationMan=4, ChildTransportationVrouw=1
14:41:10	Out	HouseholdID=2668, ChildTransportationMan=4, ChildTransportationVrouw=1
14:41:10	Out	HouseholdID=2690, ChildTransportationMan=4, ChildTransportationVrouw=1
14:41:10	Out	HouseholdID=22125, ChildTransportationMan=4, ChildTransportationVrouw=1
14:41:10	Out	HouseholdID=5486, ChildTransportationMan=4, ChildTransportationVrouw=1
14:41:10	Out	HouseholdID=31531, ChildTransportationMan=4, ChildTransportationVrouw=1
14:41:10	Out	HouseholdID=719, ChildTransportationMan=4, ChildTransportationVrouw=1
14:41:10	Out	HouseholdID=1309, ChildTransportationMan=4, ChildTransportationVrouw=1
14:41:10	Out	HouseholdID=3054, ChildTransportationMan=4, ChildTransportationVrouw=1

Figuur 46: Applicaton output project 4

Bijlage 19: Casestudie 4, resultaten Coral8

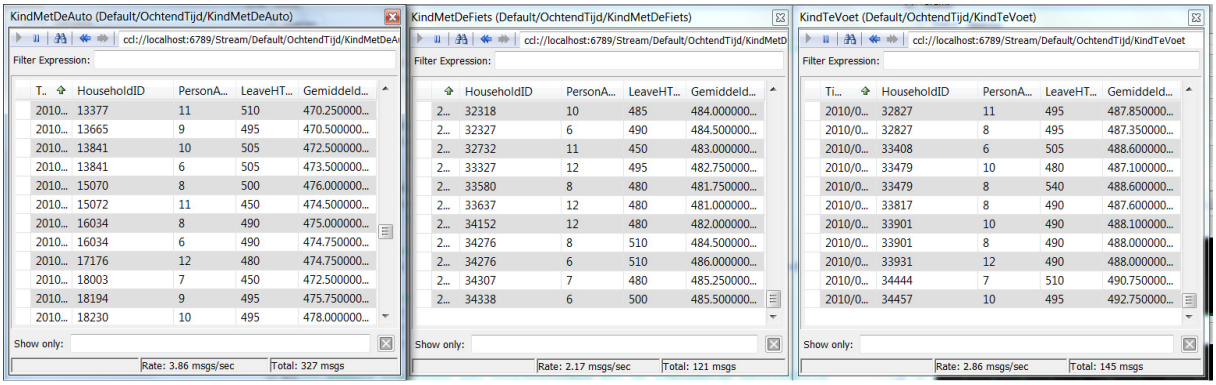


Figuur 47: Flow view project 4



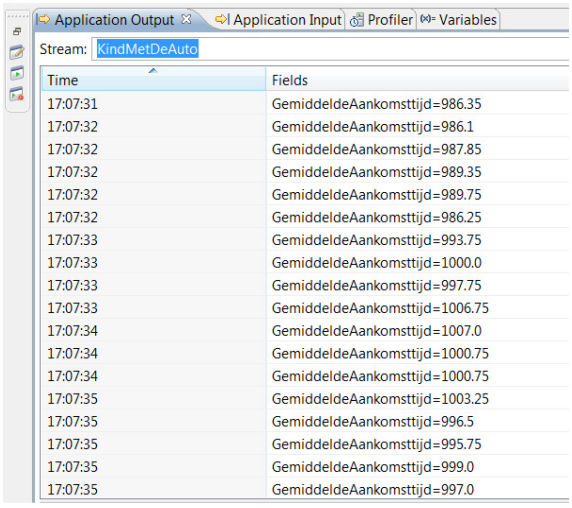
Figuur 48: Stream view project 4

Bijlage 20: Project 5, resultaten Coral8



Figuur 49: Stream view project 5

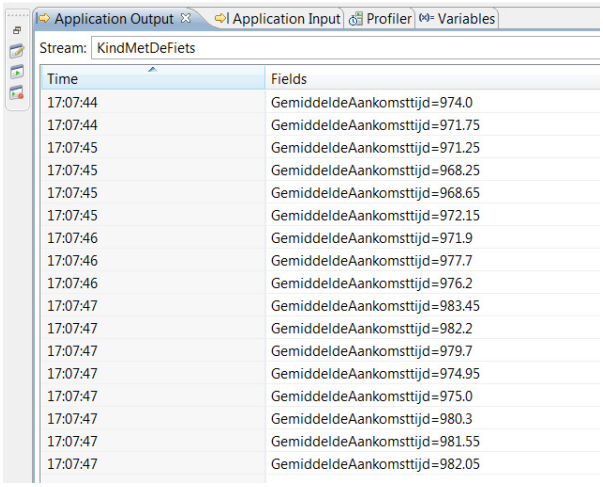
Bijlage 21: Project 5, resultaten StreamBase


Application Output | Application Input | Profiler | Variables

Stream: KindMetDeAuto

Time	Fields
17:07:31	GemiddeldeAankomsttijd=986.35
17:07:32	GemiddeldeAankomsttijd=986.1
17:07:32	GemiddeldeAankomsttijd=987.85
17:07:32	GemiddeldeAankomsttijd=989.35
17:07:32	GemiddeldeAankomsttijd=989.75
17:07:32	GemiddeldeAankomsttijd=986.25
17:07:33	GemiddeldeAankomsttijd=993.75
17:07:33	GemiddeldeAankomsttijd=1000.0
17:07:33	GemiddeldeAankomsttijd=997.75
17:07:33	GemiddeldeAankomsttijd=1006.75
17:07:34	GemiddeldeAankomsttijd=1007.0
17:07:34	GemiddeldeAankomsttijd=1000.75
17:07:34	GemiddeldeAankomsttijd=1000.75
17:07:35	GemiddeldeAankomsttijd=1003.25
17:07:35	GemiddeldeAankomsttijd=996.5
17:07:35	GemiddeldeAankomsttijd=995.75
17:07:35	GemiddeldeAankomsttijd=999.0
17:07:35	GemiddeldeAankomsttijd=997.0

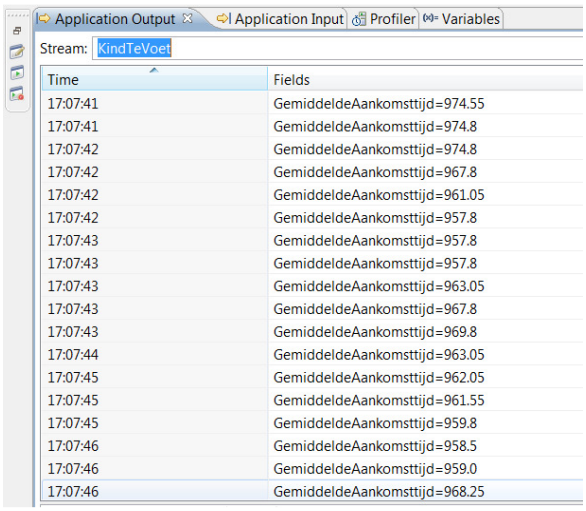
Figuur 50: Application output project 5: Kind met de auto


Application Output | Application Input | Profiler | Variables

Stream: KindMetDeFiets

Time	Fields
17:07:44	GemiddeldeAankomsttijd=974.0
17:07:44	GemiddeldeAankomsttijd=971.75
17:07:45	GemiddeldeAankomsttijd=971.25
17:07:45	GemiddeldeAankomsttijd=968.25
17:07:45	GemiddeldeAankomsttijd=968.65
17:07:45	GemiddeldeAankomsttijd=972.15
17:07:46	GemiddeldeAankomsttijd=971.9
17:07:46	GemiddeldeAankomsttijd=977.7
17:07:46	GemiddeldeAankomsttijd=976.2
17:07:47	GemiddeldeAankomsttijd=983.45
17:07:47	GemiddeldeAankomsttijd=982.2
17:07:47	GemiddeldeAankomsttijd=979.7
17:07:47	GemiddeldeAankomsttijd=974.95
17:07:47	GemiddeldeAankomsttijd=975.0
17:07:47	GemiddeldeAankomsttijd=980.3
17:07:47	GemiddeldeAankomsttijd=981.55
17:07:47	GemiddeldeAankomsttijd=982.05

Figuur 51: Application output project 5: Kind met de fiets


Application Output | Application Input | Profiler | Variables

Stream: KindTeVoet

Time	Fields
17:07:41	GemiddeldeAankomsttijd=974.55
17:07:41	GemiddeldeAankomsttijd=974.8
17:07:42	GemiddeldeAankomsttijd=974.8
17:07:42	GemiddeldeAankomsttijd=967.8
17:07:42	GemiddeldeAankomsttijd=961.05
17:07:42	GemiddeldeAankomsttijd=957.8
17:07:43	GemiddeldeAankomsttijd=957.8
17:07:43	GemiddeldeAankomsttijd=957.8
17:07:43	GemiddeldeAankomsttijd=963.05
17:07:43	GemiddeldeAankomsttijd=967.8
17:07:43	GemiddeldeAankomsttijd=969.8
17:07:44	GemiddeldeAankomsttijd=963.05
17:07:45	GemiddeldeAankomsttijd=962.05
17:07:45	GemiddeldeAankomsttijd=961.55
17:07:45	GemiddeldeAankomsttijd=959.8
17:07:46	GemiddeldeAankomsttijd=958.5
17:07:46	GemiddeldeAankomsttijd=959.0
17:07:46	GemiddeldeAankomsttijd=968.25

Figuur 52: Application output project 5: Kind te voet

Bibliografie

Aleri. (2009). *Coral8 documents*. Opgeroepen op December 5, 2009, van Sybase Aleri: http://www.aleri.com/developers/documents/coral8

Arasu, A., & Widom, J. (n.d.). *Resource sharing in continuous sliding-window aggregates.* Opgeroepen op 30 oktober, 2009, van vldb: http://www.vldb.org/conf/2004/RS9P2.PDF

Arasu, A., Babcock, B., Babu, S., Cieslewicz, J., Datar, M., Ito, K., et al. (n.d.). *Stream: The Stanford Data Stream Management System.* Opgeroepen op oktober 10, 2009, van Infolab Stanford University: http://infolab.stanford.edu/~usriv/papers/streambook.pdf

Babcock, B., Babu, S., Datar, M., Motwani, R., & Widom, J. (n.d.). *Models and issues in data stream systems.* Opgeroepen op februari 10, 2009, van Infolab: http://infolab.usc.edu/csci599/Fall2002/paper/DML2_streams-issues.pdf

Babu, S., & Widom, J. (2001). *Continuous queries over data streams.* Opgeroepen op oktober 15, 2009, van Stanford university Infolab: http://ilpubs.stanford.edu:8090/527/

Bergamini, A., Biersack, E. W., Goebel, V., Plagemann, T., Tolu, G., & Urvoy-Keller, G. (2004). *Using data stream management systems for traffic analysis - A case study.* Opgeroepen op oktober 31, 2009, van PAM: http://www.pam2004.org/papers/113.pdf

Chakravarthy, S., & Jian, Q. (2009). *Stream data processing: a quality of service perspective.* New York, USA: Springer.

Chandrasekaran, S., & Franklin, M. J. (n.d.). *Streaming Queries over Streaming Data.* Opgeroepen op oktober 30, 2009, van VLDB Endowment Inc.: http://www.vldb.org/conf/2002/S07P01.pdf
Chapple, M. (n.d.). *SQL fundamentals.* Opgeroepen op oktober 28, 2009, van About.com: databases.

FPS economie - Algemene directie statistiek en economische informatie. (2001). *Algemeen socio-economische enquête 2001.* Opgeroepen op april 15, 2010, van http://economie.fgov.be/nl/binaries/pr140_nl%5B1%5D_tcm325-66383.pdf

Goedertier, S. (2008, december 16). *Event logs, de zwarte doos van onze bedrijfsprocessen.* Opgeroepen op oktober 10, 2009, van ITProfessional: http://www.itprofessionals.be/print.cfm?id=96050

Gualtieri, M., & Rymer, J. R. (2009). *The Forrester wave: complex event processing (CEP) platforms Q3, 2009.* Opgeroepen op oktober 10, 2009, van Forrester: http://www.cnetdirectintl.com/direct/fr/2009/progress/0907_centre_ressources/ressources/news/Report_wave_complex_event_processing_cep_platforms.pdf

Hannes, E., Lui, F., Vanhulsel, M., Janssens, D., Bellemans, T., Vanhoof, K., et al. (2010). Trackin household routines using scheduling hypothesis embedded in skeletons. België.

Jones, T., & W, S. (2009, december 1). *StreamBase Named 2010 World Economic Forum Technology Pioneer.* Opgeroepen op februari 5, 2010, van StreamBase: http://streambase.typepad.com/streambase_stream_process/2009/12/streambase-wins-davos.html

McReynolds, S. (2007, September). *Complex event processing in the real world.* Opgeroepen op oktober 10, 2009, van Oracle: http://www.oracle.com/technologies/soa/docs/oracle-complex-event-processing.pdf

Michelson, B. M. (2006, februari 2). *Event-driven architecture overview.* Opgeroepen op oktober 17, 2009, van http://soa.omg.org/Uploaded%20Docs/EDA/bda2-2-06cc.pdf
(n.d.). Opgeroepen op november 2009, van StreamBase: www.streambase.com

Palmer, M. (2010, februari 4). *What does the Sybase/Aleri acquisition means? .* Opgeroepen op februari 5, 2010, van StreamBase:

http://streambase.typepad.com/streambase_stream_process/2010/02/sybase-aleri-acquisition-what-does-it-mean.html

Pieciukiewicz, T., Stencel, K., & Subieta, K. (n.d.). *Recursive Query Processing in SBQL.* Opgeroepen op mei 15, 2009, van Object Database management Systems: http://www.odbms.org/download/030.03%20Subieta%20Recursive%20Query%20Processing%20in%20SQBL%20March%202008.PDF

Purich, P. (2009). *Oracle CEP - CQL Language Reference.* Opgeroepen op oktober 10, 2009, van Oracle: http://download.oracle.com/docs/cd/E12839_01/doc.1111/e12048/intro.htm

Purich, P. (2009c). *Oracle CEP-EPL language reference.* Opgeroepen op oktober 1, 2009, van Oracle: http://download.oracle.com/docs/cd/E13157_01/wlevs/docs30/epl_guide/overview.html

Purich, P. (2009b, mei). *Oracle CEP-Getting started.* Opgeroepen op oktober 10, 2009, van Oracle: http://download.oracle.com/docs/cd/E12839_01/doc.1111/e14476.pdf

Schouw, B. (n.d. b). *Complex event processing, het belang van events.* Opgeroepen op oktober 28, 2009, van Kennisportal: http://www.kennisportal.com/main.asp?ChapterID=6603

Schouw, B. (n.d. a). *White paper complex event processing.* Opgeroepen op februari 5, 2010, van Kennisportal: http://www.kennisportal.be/main.asp?ChapterID=6587

Segers, I. (2007, Augustus 16). *Investigating the application of process mining for auditing purposes.* Opgeroepen op oktober 10, 2009, van Bibliotheek TU/e: http://alexandria.tue.nl/extra2/afstversl/tm/Segers2007.pdf

Siegele, L. (2002, februari 1). *The real-time economy: how about now?* Opgeroepen op oktober 24, 2009, van CFO: http://www.cfo.com/article.cfm/3003286

StreamBase. (2010, februari 4). *StreamBase Offers Amnesty to Sybase-Aleri-Coral8 Customers .* Opgeroepen op februari 5, 2010, van StreamBase: http://www.streambase.com/0d370556-e9a3-4415-9228-d11fd93ad1cf/press-release-detail.htm

StreamBase Systems. (2010). Help-StreamBase Studio 6.5.

Auteursrechtelijke overeenkomst

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Querying event logs

Richting: **master in de toegepaste economische wetenschappen : handelsingenieur in de beleidsinformatica**
Jaar: **2010**

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Voor akkoord,

Swinnen, Jo

Datum: **1/06/2010**