

# Tech Talk: Building SQL Statements

“EBR Applications” are user interfaces that builds SQL statements for retrieving information from the database.

## Other SQL Tools:

- Oracle – SQL Developer
- SQL Server Management Studio
- AnySQL Maestro
- TOAD

# Understanding Databases

## Databases

Collection of Tables designed to combine data into useful information

## Tables

Defined structures for capturing and storing data – “Employee\_Master”

## Fields

Column(s) of data defined within a Table – “Store, Cashier, Cashier Name...”

## Records

Row(s) of data within a table that is the collection of cells – “12, 342, John Smith ...”

## Cells

Storage space for data elements

## Data Element

Information stored in a single cell – “342”

## Employee\_Master Data

Store	Cashier	Cashier Name	Address	City	State	HireDate	TermDate
12	342	John Smith	45 Main Street	Madison	WI	2/3/2008	5/5/2008
12	356	Lucy Bean	1874 Board Street	Madison	WI	4/1/2008	
12	378	Joe Franklin	65 Staple Road	Madison	WI	5/6/2008	
115	409	George Bell	376 Three Mile Road	Cleveland	OH	8/14/2008	
115	425	Brock Tanner	90 River View Place	Cleveland	OH	10/8/2008	11/9/2009
115	462	Ann Bridge	7771 Hartley Court	Akron	OH	2/2/2009	
231	353	Shane Holiday	911 Knot Tree Circle	Stamford	CT	3/15/2008	
231	675	Jill McDonald	2131 Uptown Ave	Stamford	CT	11/8/2009	

# Basic Database

## A Generic Database for a Typical EBR System

### Tables and Fields Structure

Store_Master
Store
StoreName
Address
City
State
StoreType
Region
District
LPRegn
LPDist

Employee_Master
Store
Cashier
Cashier Name
Address
City
State
HireDate
TermDate

Trans_Header
Store
Date
Time
Register
Transaction
Cashier
Net_Qty
Net_PLU_Amt
Net_Disc_Amt
Net_Actl_Amt
Net_Tndr_Amt
EmpSale
EmpDiscNum
Cash Flag
Check Flag
Credit Flag
Gift Card Flag

Trans_Details
Store
Date
Time
Register
Transaction
Cashier
RecordType
LineVoid
ItemNo
Qty
PLU_Amt
Actual_Amt
Discount_Amt
Trans_Details_Amt
Trans_DetailsType
Account
EmpSale
EmpDiscNum
Orig_Store
Orig_Date
Orig_Register
Orig_Trans

# Basic SQL Query

**Select** table1.<field1>,  
table1.<field2>, ...,  
table1.<fieldX>  
**From** <table1>

Select <field1>, <field2>, ..., <fieldX>  
From <table1>  
**Where** <fieldX> = <a value>  
**Order by** <fieldX>

“Select” – which fields does one want to see  
“From” – which table(s) is the data stored

“Where” – Filters the Record that match  
the criteria given in the SQL  
“Order by” – Sorts the data by the field(s)

Select Store, Cashier, CashierName  
From Employee\_Master

Select Store, Cashier, CashierName  
From Employee\_Master  
Where Store = 12  
Order by Cashier

Store	Cashier	Cashier Name
12	342	John Smith
12	356	Lucy Bean
12	378	Joe Franklin
115	409	George Bell
115	425	Brock Tanner
115	462	Ann Bridge
231	353	Shane Holiday
231	675	Jill McDonald

Store	Cashier	Cashier Name
12	342	John Smith
12	356	Lucy Bean
12	378	Joe Franklin

**Note:** Table Name can be dropped when there is only one table.

# Comparison and Logical Operators

Operators are used to filter the data or narrow the SQL Query Results

## Comparison:

- = - Equal to
- <> - Not Equal to
- > - Greater Than
- >= - Greater Than or Equal to
- < - Less Than
- <= - Less Than or Equal to

- In – Multiple Items Equal to; i.e. Store in (12,115)
- Between – data within a range; i.e. Cashiers between 300 and 350
- Like – Wildcard searches “\_” single position “%” multiple position;  
i.e. Name like ‘Jo%’ or Address like ‘9\_\_ Knot Tree Circle’
- NULL – looks for blank or no data in the cell is NULL or is not NULL;  
i.e. Term\_Date is Null

## Logical:

- AND – All Comparisons have to match
- OR – One or More Comparisons have to match
- NOT – Reverse of Comparison result

Select Store, Cashier, CashierName  
From Employee\_Master  
Where Store in (12, 115)

Store	Cashier	Cashier Name
12	342	John Smith
12	356	Lucy Bean
12	378	Joe Franklin
115	409	George Bell
115	425	Brock Tanner
115	462	Ann Bridge

Select Store, Cashier, CashierName  
From Employee\_Master  
Where Store in (12, 115) and CashierName like '%B%'

Store	Cashier	Cashier Name
12	356	Lucy Bean
115	409	George Bell
115	425	Brock Tanner
115	462	Ann Bridge

# Aggregate Query with Calculation Functions

Aggregate - “formed by the conjunction or collection of particulars into a whole mass or sum; total; combined” –

[www.dictionary.com](http://www.dictionary.com)

```
Select <field1>, Sum(<field2>), ..., Avg(<fieldX>)
From <table1>
Where <fieldX> = <a value>
Group by <field1>, ..., <fieldX>
Having Sum(<field2>) > <a value>
```

Group by – Columns that data should be summarized for; i.e. Sales by “**Store & Cashier**”  
Having - Filters the Results that match the criteria given in the SQL

## Functions that total or combine records that are similar:

Count(), Sum(), Avg()  
Min(), Max()

# Aggregate Query Example

Select Store, City, Count(Cashier) “# of Cashiers”  
From Employee\_Master  
Where TermDate is Null  
Group by Store, City

Store	City	# of Cashiers
12	Madison	2
115	Cleveland	1
115	Akron	1
231	Stamford	2

Select Store, City, Count(Cashier) “# of Cashiers”  
From Employee\_Master  
Group by Store, City  
Having Count(Cashier) >=2

Store	City	# of Cashiers
12	Madison	3
115	Cleveland	2
231	Stamford	2

Store	Cashier	Cashier Name	Address	City	State	HireDate	TermDate
12	342	John Smith	45 Main Street	Madison	WI	2/3/2008	5/5/2008
12	356	Lucy Bean	1874 Board Street	Madison	WI	4/1/2008	
12	378	Joe Franklin	65 Staple Road	Madison	WI	5/6/2008	
115	409	George Bell	376 Three Mile Road	Cleveland	OH	8/14/2008	
115	425	Brock Tanner	90 River View Place	Cleveland	OH	10/8/2008	11/9/2009
115	462	Ann Bridge	7771 Hartley Court	Akron	OH	2/2/2009	
231	353	Shane Holiday	911 Knot Tree Circle	Stamford	CT	3/15/2008	
231	675	Jill McDonald	2131 Uptown Ave	Stamford	CT	11/8/2009	

# Table and Column Alias

```
Select t1.<field1>, Sum(t1.<field2>) "New Name", ..., t1.<fieldX>  
From <table1> t1
```

Alias are used to define the tables with shorter name.

Alias can be used to give the data results a new name for the column header.

```
Select emp.Store, emp.City, Count(emp.Cashier) "# of Cashiers"  
From Employee_Master emp  
Group by emp.Store, emp.City  
Having Count(emp.Cashier) >=2
```

Store	City	# of Cashiers
12	Madison	3
115	Cleveland	2
231	Stamford	2

Remember Note from Slide 4 : Table can be dropped when there is only one table

# Table Joins

The key to databases is the ability to capture and store data with little to no duplication, especially for large data sets, or to link data from different data sources.

Duplication of Data:

Names, Descriptions & Addresses

Linking Data for different sources:

Store, Employee, Customer, Item/Product & Inventory/Cycle Count Information

## SQL Statement for Joins:

Select t1.<field1>, t2.<field2>, ..., t3.<fieldX>

From <table1> t1, <table2> t2, <table3> t3

Where t1.<fieldN> = t2.<field2> and

t1.<fieldN> = t3.<field1> and t1.<fieldP> = t3.<field2>

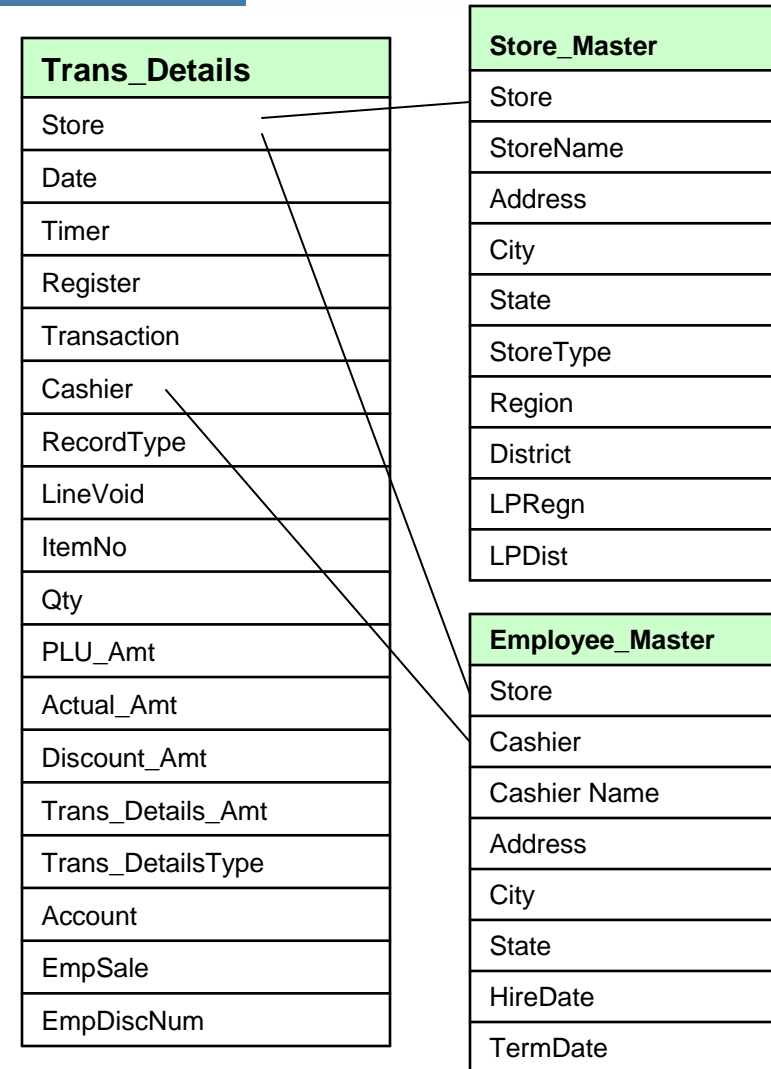
# Table Joins – Inner or “Exact Match”

A join requires two tables to have one or more fields of data that match to allow for combining the data from the two tables (Field Name doesn't have to match but data type does).

```
Select trans.Store, str.StoreName,  
trans.Cashier, emp.CashierName  
trans.date, trans.time  
trans.register, trans.transaction  
... trans.EmpSale
```

```
From Trans_Details trans,  
Store_Master str  
Employee_Master emp
```

```
Where trans.store = str.store and  
trans.store = emp.store and  
trans.cashier = emp.cashier
```



# Table Joins – Outer (Left or Right)

Outer Joins are the ability to link two tables where all records from one table are included in the results even if there is no match to the second table. If there is no match the data from the second table will be left blank or null.

Left vs. Right depends on order the table are typed into SQL and which one all records needs to be selected from.

## EXAMPLE: (Invalid Original Receipt)

Select dtls.Store, dtls.Date, ..., hdr.Cashier  
"Orig\_Cashier"

**From Trans\_Details dtls**

**Left Outer Join Trans\_Header hdr**

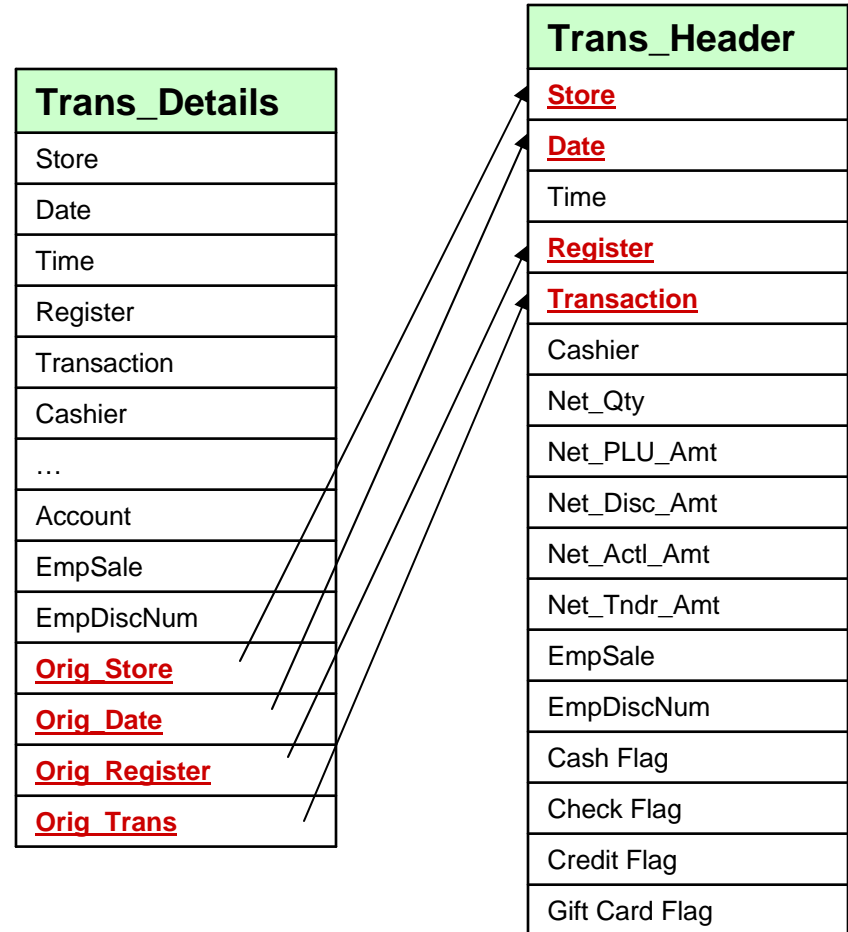
dtls.Orig\_Store = hdr.Store and

dtls.Orig\_Date = hdr.Date and

dtls.Orig\_Register = hdr.Register and

dtls.Orig\_Trans = hdr.Transaction

**Where hdr.Cashier is NULL**



# Case Statements

Case statements is SQL's "IF":

If EmpSale is "Y" then 1 else 0

If SwipedFlag is "Y" then 1 else 0

---

Case <fieldname>  
When <testvalue>  
Then <returnvalue1>  
Else <returnvalueX>  
End

Case EmpSale when 'Y'  
Then 1 Else 0 End

Case SwipedFlag when 'Y'  
Then 1 Else 0 End

---

**# of Emp Sales:** **Sum**(Case EmpSale when 'Y' Then 1 Else 0 End)  
or **Sum**(**Decode**(EmpSales,'Y',1,0)) for Oracle Users

# Putting It All Together

## SQL Example:

```
SELECT Trans_Header.DIVISION,  
Trans_Details.ACCOUNT,  
Trans_Details.ACCOUNTNUM_HASH,  
MAX(STOREMST.LPDISTRICT) "Max RLPM",  
COUNT(DISTINCT(Trans_Details.STORENUM)) "# of Stores",  
SUM(DECODE(Trans_Details.EMPSALE,'Y',1,0)) "# of Emp Trans",  
MAX(Trans_Details.EMPNUM) "Max Employee",  
(SUM(DECODE(Trans_Header.GIFTCARD,'2',1,'3',1,0)) + SUM(DECODE(Trans_Header.GIFTCERT,'2',1,'3',1,0))) "# of Gift Cd/Cert Rdmptns",  
SUM(DECODE(Trans_Header.STORECREDIT,'2',1,'3',1,0)) + SUM(DECODE(Trans_Header.MERCHCARD,'2',1,'3',1,0)) "# of Merch Cd/Cert Rdmptns",,  
  
FROM Trans_Details, Trans_Header, STOREMST  
  
WHERE Trans_Details.storenum = Trans_Header.storenum (+) and Trans_Details.transdate = Trans_Header.transdate (+) and  
Trans_Details.regnum = Trans_Header.regnum (+) and Trans_Details.transnum = Trans_Header.transnum (+) AND  
storemst.store = Trans_Details.storenum and storemst.division = Trans_Details.division  
AND  
Trans_Details.TRANSDATE between <Start Date> and <End Date>  
AND (Trans_Details.RECTYPE = 'TND' AND  
Trans_Details.RECCODE IN ('09','30','31','32','33')  
  
GROUP BY Trans_Details.STR9, Trans_Details.ACCOUNTNUM_HASH, Trans_Header.DIVISION  
  
HAVING SUM(DECODE(Trans_Details.EMPSALE,'Y',1,0)) > 0 AND  
((SUM(DECODE(Trans_Header.GIFTCARD,'2',1,'3',1,0)) +  
SUM(DECODE(Trans_Header.GIFTCERT,'2',1,'3',1,0))) > 1 OR  
(SUM(DECODE(Trans_Header.STORECREDIT,'2',1,'3',1,0)) +  
SUM(DECODE(Trans_Header.MERCHCARD,'2',1,'3',1,0))) > 0)
```

**Views and Indexes should be handled by a Database Administrator (DBA) but as a database (or EBR) user, one should understand the capabilities and the ease of creating them.**

**“Views” are typically virtual tables that are built on top of other Tables or Views but doesn’t store data but store the commands (SQL) to retrieve the data.**

**Create View V\_<view name>**

**As** Select t1.<field1>, Sum(t1.<field2>) “Amount”, ..., t1.<fieldX>  
From <table1> t1

**Indexes help SQL Statements run faster by creating a list that tracks where the data is stored to prevent full table scans. An example would be on an account number search, it would know to only search between records 100 and 300 instead 1 to 12M.**

**Create Index “Index\_Fields\_Name”**

**On** <table1> (<field1>, <field2>, ... <fieldN>)

## Basic SQL (Instructions)

<http://www.firstsql.com/tutor.htm>

## Functions (Formulas)

<http://www.1keydata.com/sql/sqlselect.html>

<http://www.sql-ref.com/> - Differences between Oracle and SQL Server

Distinct(), Count(), Sum(), Avg()  
Abs(), Min(), Max(),  
DateDiff(), GetDate() or Sysdate(),  
Substring(), Left(), Right(), Length() or Len(), Concatenate()  
To\_char(), To\_Number(), To\_Date() or Cast()  
Trim(), LTrim(), RTrim()

[www.google.com](http://www.google.com) is your friend for research!!