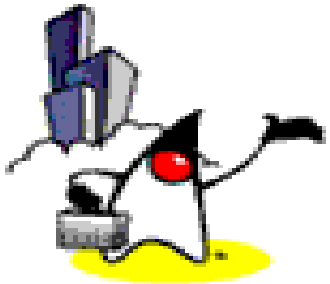
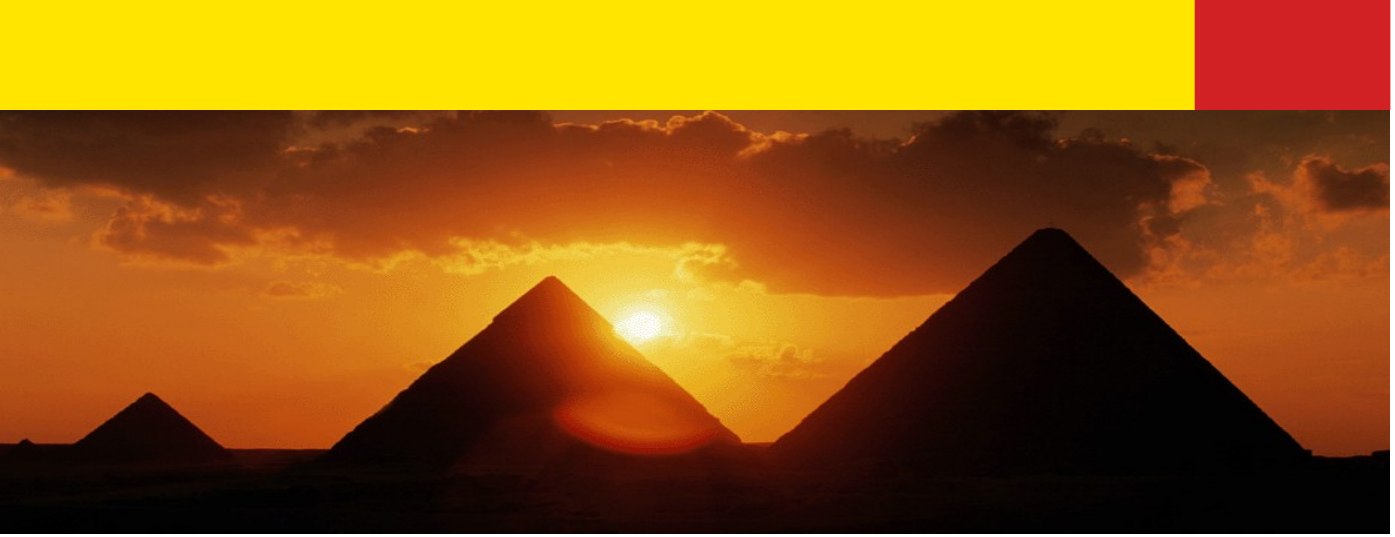
SQL

(Structured Query Language)

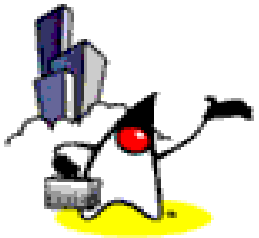


Agenda

- What is SQL?
- SQL Terminology
- Creating and Deleting Tables
- Select Statement for querying
- Join Statement
- Aliases
- Sub-query
- Comparison, Character match, Numerical expression
- Insert, Update, Delete
- SQL Data-types



What is SQL?



What is SQL?

- SQL provides a way to retrieve and manipulate data in a relational database
- SQL is used for all relational database operations, including administration, schema creation and data retrieval
  - Data definition
  - Data manipulation

What SQL IS Not

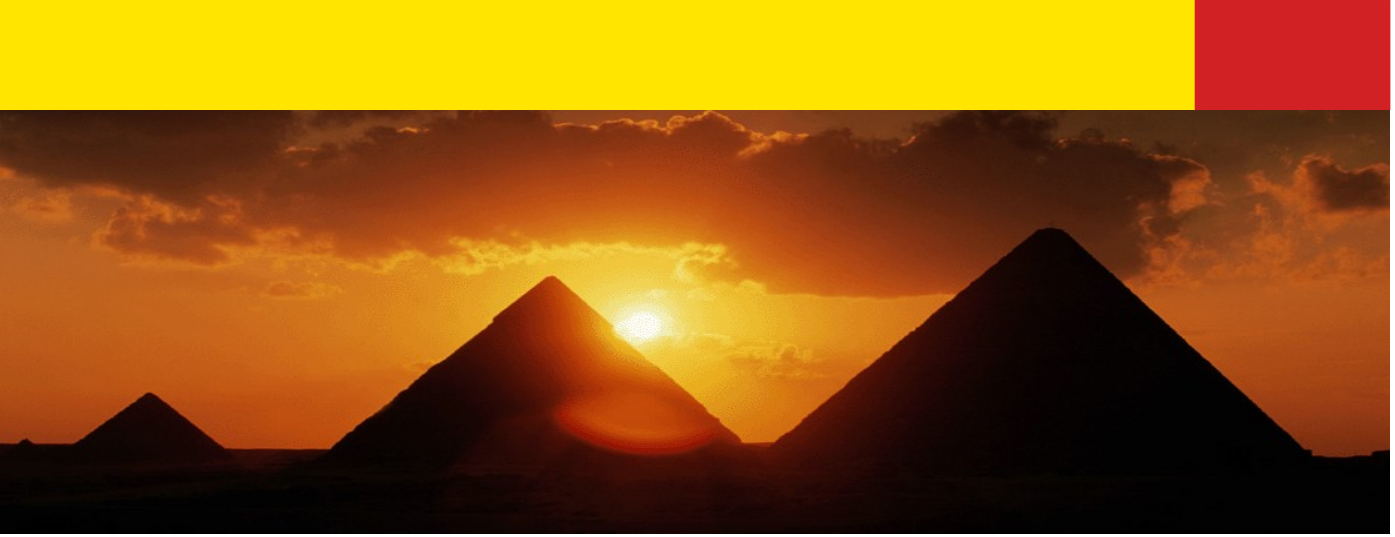
- SQL is a not a programming language
  - It does not provide any flow-of-control programming constructs, function definitions, do-loops, or if-then-else statements
  - Unlike modern programming languages that enable you to define a data type for a specific purpose, SQL forces you to choose from a set of predefined data types when you create or modify a column.
- SQL is not a procedural language
  - You use SQL to tell the database what data to retrieve or modify without telling it how to do it

Similarity with Programming Language

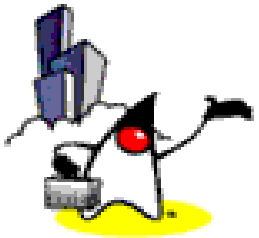
- They both usually give you more than one way to accomplish the same goal
  - Various SQL statements may achieve the same results (but differ in processing efficiency and clarity of code)

Case Sensitivity

- SQL is not case sensitive
  - You can mix uppercase and lowercase when referencing SQL keywords (SELECT, INSERT, ...), table names and column names
  - However, case does matter when referring to the contents of a column; if you ask for all the store's where the store name is lower case, but in the database all the names are stored in upper case, you wont retrieve any rows at all



SQL Terminology

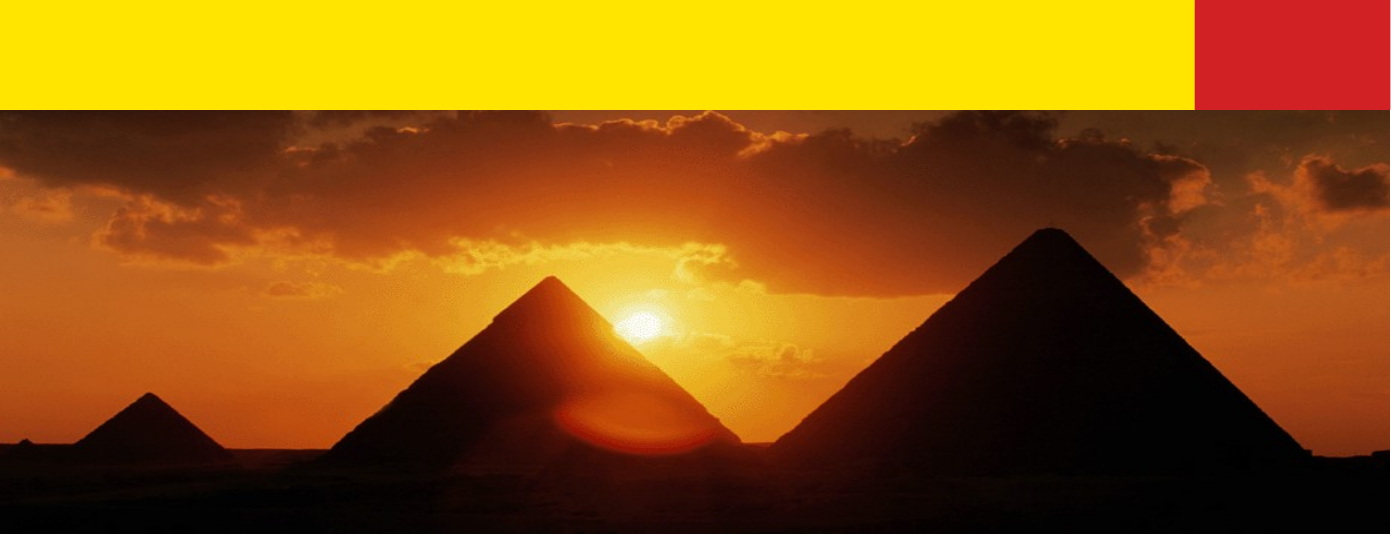


SQL Terms

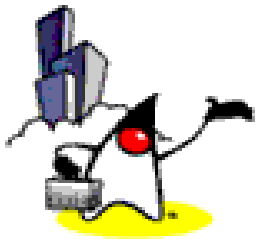
- Table
  - A set of rows
  - Analogous to a “file”
- Column
  - A column is analogous to a “field” of a record
  - Each column in a given row has a single value

SQL Terms (Continued)

- Row
  - Analogous to a “record” of a file
  - All rows of a table have the same set of columns
- Primary Key
  - One of more columns whose contents are unique within a table and thus can be used to identify a row of that table



Creating & Deleting Tables and other Operations via DDL

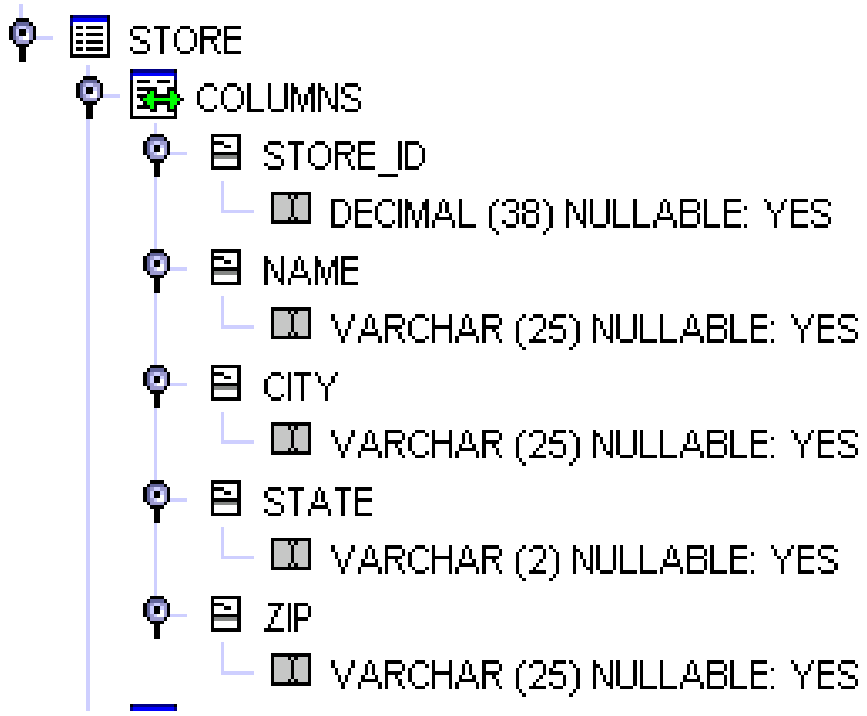


Data Definition Language (DDL)

- DDL is used for defining the database structure
  - Create Table, Alter Table, Drop table
  - Create Index, Drop Index
  - Create User, Alter User, Drop User
  - Create Role, Drop Role
  - Create Schema, Drop Schema
  - Create View, Drop View
  - Create Trigger, Drop Trigger

CREATE TABLE

```
create table store
(store_id NUMERIC,
 name VARCHAR(25),
 city VARCHAR(25),
 state VARCHAR(2),
 zip VARCHAR(25));
```



ALTER TABLE

```
ALTER TABLE store ADD UNIQUE (C1);
ALTER TABLE store ADD ORDER_NUM INT;
ALTER TABLE store ADD CONSTRAINT
   constraint_0 FOREIGN KEY (C1)
   REFERENCES T1 (C1);
ALTER TABLE store DROP ORDER_NUM
   CASCADE;
ALTER TABLE store RENAME TO T1;
```

DROP TABLE

```
DROP TABLE store;
```

CREATE INDEX & DROP INDEX

```
CREATE  INDEX  customer_idx
 ON "PBPUBLIC"."CUSTOMER_TBL"
 ( "CUSTOMER_NUM" ASC ,"NAME" ASC  )

DROP INDEX ORDER_TBL.ORDER;
```

CREATE/ALTER/DROP USER

```
CREATE USER PoInT PASSWORD BaSE;
CREATE USER "PoInT" PASSWORD "BaSE";

ALTER USER Scott PASSWORD lion;
ALTER USER Scott DEFAULT ROLE CEO;

DROP USER ENGINEERING_MANAGER
   CASCADE;
```

CREATE/DROP VIEW

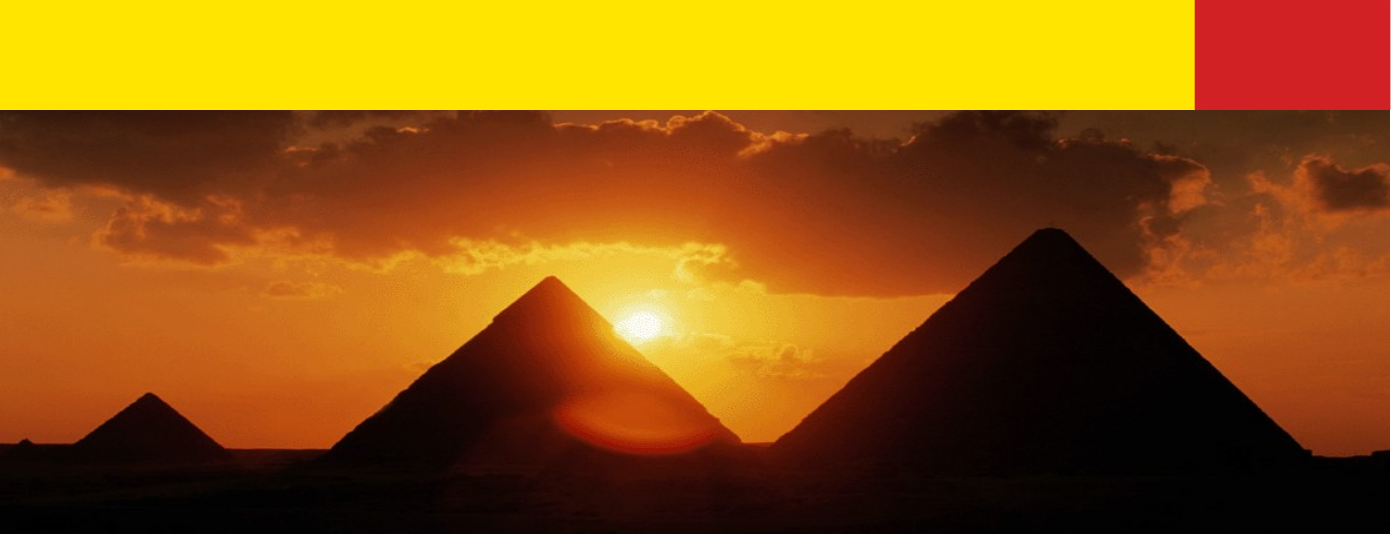
```
CREATE VIEW customer_order
AS select
   order_num,order_tbl.customer_num,customer
   _tbl.name
FROM order_tbl,customer_tbl
WHERE product_num = 10;

DROP VIEW customer_order;
```

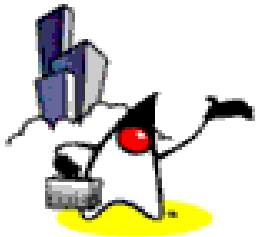
CREATE/DROP TRIGGER

```
CREATE TRIGGER trigger2
BEFORE UPDATE ON product_tbl
REFERENCING NEW AS NEWROW
FOR EACH ROW
WHEN (NEWROW.qty_on_hand < 0)
SET NEWROW.qty_on_hand = 0;

DROP TRIGGER trigger2
```



Select Statement



SELECT Statement

SELECT [DISTINCT | ALL]
  column_expression1, column_expression2, ...
  [FROM from_clause]
  [WHERE where_expression]
  [GROUP BY expression1, expression2, ...]
  [HAVING having_expression]
  [ORDER BY {*column* | *expression*} *[asc* | *desc]]*

- **distinct** keyword eliminates duplicates

SELECT Statement

Select * from customer_tbl;

CUSTOMER_NUM	DISCOUNT_CODE	ZIP	NAME	ADDR_LN1
1	N	33015	SuperCom	490 Rivera Drive
2	M	33055	Livingston Enterprises	9754 Main Street
25	M	75200	Oak Computers	8989 Qume Drive
3	L	12347	MicroApple	8585 Murray Drive
36	H	94401	HostProCom	65653 El Camino
106	L	95035	CentralComp	829 Flex Drive
149	L	95117	Golden Valley Computers	4381 Kelly Ave
863	N	94401	HPSystems	456 4th Street
777	L	48128	West Valley Inc.	88 North Drive
753	H	48128	Ford Motor Co	2267 Michigan Ave
722	N	48124	Small Car Parts	52963 Outer Dr
409	L	10095	NY Media Productions	4400 22nd Street

FROM Clause

- Comma delimited list of tables to retrieve data from
  - With or without aliases

WHERE Clause

- Determines exactly which rows are retrieved
- Qualifications in the where clause
  - Comparison operators (=,>,<, >=,<=)
  - Ranges (between and not between)
  - Character matches (like and not like)
  - Unknown values (is null and is not null)
  - Lists (in and not in)
  - Combinations of the above (and, or)

WHERE Clause (Continued)

- Not negates any Boolean expressions and keywords such as like, null, between and in

Example: WHERE Clause

```
select Customer_num, discount_code, zip, name
 from customer_tbl
where discount_code in ('N','M','L')
    and zip like '33%'
```

CUSTOMER_NUM	DISCOUNT_CODE	ZIP	NAME
2	M	33055	Livingston Enterprises
1	N	33015	SuperCom

GROUP BY Clause

- The group by function organises data into groups based on the contents of a column(s)
  - Usually used with an aggregate function in the select list
  - The aggregate is calculated for each group
  - All null values in the group by column are treated as one group
- Grouping can be done by a *column_name* or by any expression that does not contain an aggregate function

GROUP BY Clause Cont.

- The group by function usually contains all columns and expressions in the select list that are not aggregates
- In conjunction with the where clause, rows are eliminated before they go into groups
- Applies a condition on a table before the groups are formed

Example: GROUP BY Clause

```
select customer_num,
       count(customer_num),
       sum(quantity)
from order_tbl
group by customer_num
```

CUSTOMER_NUM	count(customer_num)	sum(quantity)
1	2	110
2	2	33
3	1	10
36	2	180
106	1	500
149	1	1000
409	1	50
410	1	100
722	1	250
753	1	100
777	1	75
863	1	100

HAVING Clause

- Set conditions on groups
  - Restricts groups
- Applies a condition to the groups after they have been formed
- having is usually used in conjunction with an aggregate function

Example: HAVING Clause

```
select product_num,
       quantity,
       shipping_cost
 from order_tbl
group by product_num,
       quantity,
       shipping_cost
having shipping_cost >
    250
```

PRODUCT_NUM	QUANTITY	SHIPPING_COST
980001	10	449
980001	50	2000.99
980005	8	359.99
980025	25	275
980030	10	275
980031	100	700
980032	100	459
980500	250	2500
986420	1000	700
988765	500	265

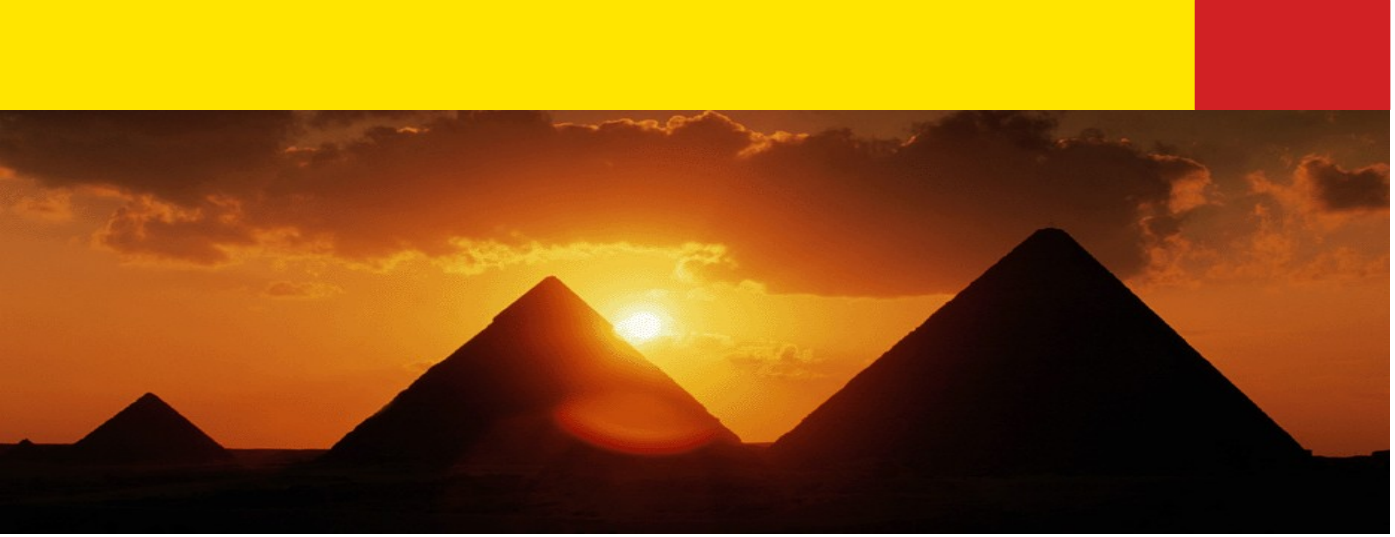
ORDER BY Clause

- The order by clause sorts the query results (in ascending order by default)
- Items named in an order by clause need not appear in the select list
- When using order by, nulls are listed first

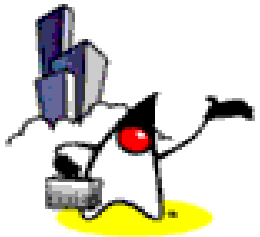
ORDER BY Clause Example

```
select product_num,
       quantity,
       shipping_cost
 from order_tbl
group by product_num,
       quantity,
       shipping_cost
having shipping_cost > 250
order by quantity,
       shipping_cost;
```

PRODUCT_NUM	QUANTITY	SHIPPING_COST
980005	8	359.99
980030	10	275
980001	10	449
980025	25	275
980001	50	2000.99
980032	100	459
980031	100	700
980500	250	2500
988765	500	265
986420	1000	700



Join Statement



Join Statement

- Retrieves data from two or more tables
- Combines tables by matching values from rows in each table
- Joins are done in the where clause
- Columns in join don't have to be in select

Join Statement (Continued)

- If you do not specify how to join rows from different tables, the database server assumes you want to join every row in each table
  - Cartesian Product
    - Very costly (cpu time)
    - May take a while to return large result set
- Column names that appear in more then one table should be prefixed by table name

Example: Join Statement

```
select c.name,
        o.quantity,
        o.shipping_cost
from order_tbl o,
        customer_tbl c
where o.customer_num =
    c.customer_num;
```

NAME	QUANTITY	SHIPPING_COST
SuperCom	10	449
SuperCom	100	459
Livingston Enterprises	8	359.99
Livingston Enterprises	25	275
MicroApple	10	275
HostProCom	120	65
HostProCom	60	55
CentralComp	500	265
Golden Valley Computers	1000	700
HPSystems	100	25
West Valley Inc.	75	105
Ford Motor Co	100	200.99
Small Car Parts	250	2500

Outer Joins

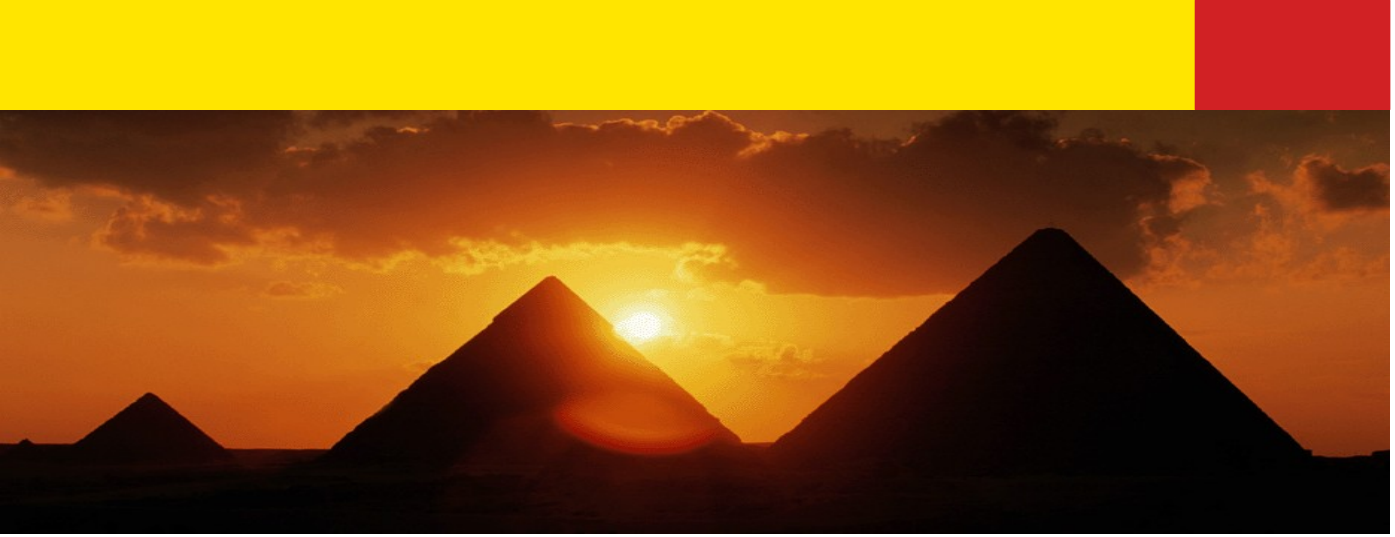
- With a join, if one row of a table is unmatched, row is omitted from result table.
- The outer join operations retain rows that do not satisfy the join condition
  - List branches and properties that are in same city along with any unmatched branches.

```
SELECT b.*, p.*
FROM branch1 b LEFT JOIN
property_for_rent1 p ON b.bcity = p.pcity;
```

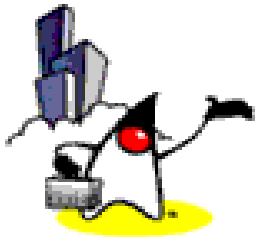
Outer Join cont.

bno	bcity	pno	pcity
B3	Glasgow	PG4	Glasgow
B4	Bristol	NULL	NULL
B2	London	PL94	London

- There is a LEFT JOIN and a RIGHT JOIN
- These can be very vendor specific (check with your vendor for outer join specifications)
- Oracle uses (+) for outer joins, MS SQL Server uses LEFT OUTER JOIN



Alias



Alias

- Use of temporary names for tables within a query
- Can be used anywhere in query
- Reduces amount of typing

ex. select pub_name, title_id
from titles t, publishers p
where t.pub_id = p.pub_id

Example: Alias

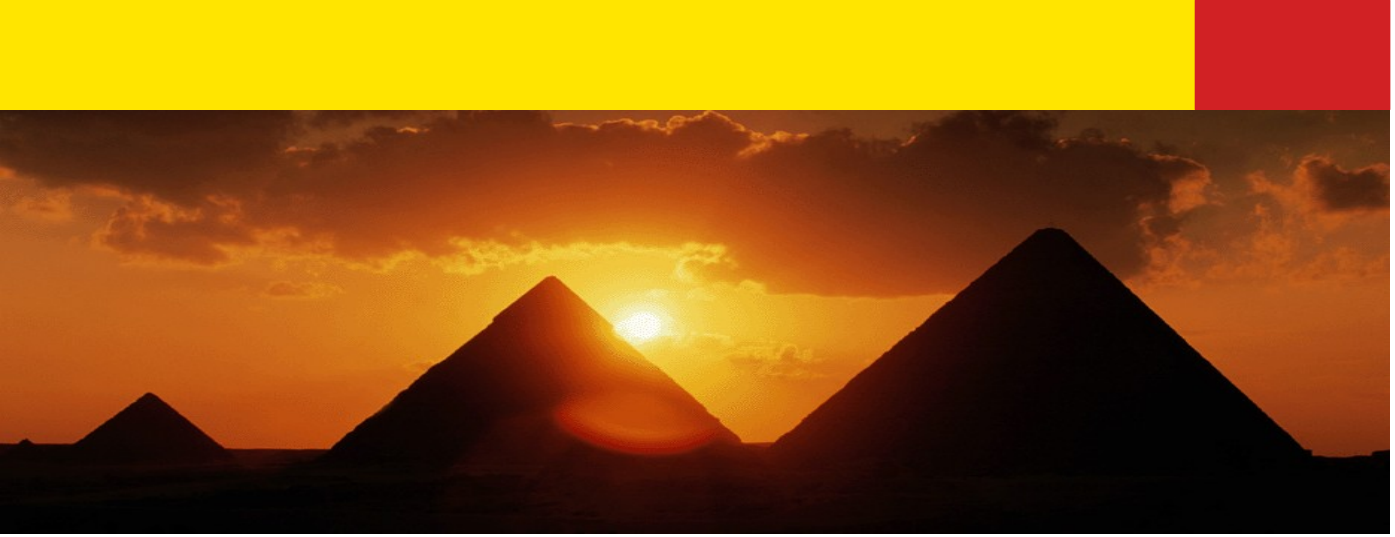
```
select type_code,
       city,
       region
from office_tbl;
```

TYPE_CODE	CITY	REGION
A	Miami	Southern
R	Atlanta	Southern
R	San Mateo	Western
W	San Francisco	Western
R	San Diego	Western
R	Oakland	Western
R	Detroit	Northern
R	New York	Eastern
R	San Francisco	Western

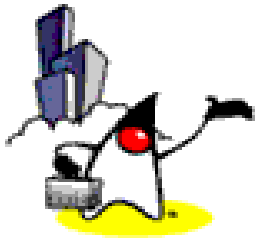
Example: Alias cont.

```
select t.description,
       o.city,
       o.region
from office_tbl o,
     office_type_code_tbl t
where o.type_code =
     t.type_code;
```

DESCRIPTION	CITY	REGION
Admin	Miami	Southern
Retail	Atlanta	Southern
Retail	San Mateo	Western
Wholesale	San Francisco	Western
Retail	San Diego	Western
Retail	Oakland	Western
Retail	Detroit	Northern
Retail	New York	Eastern
Retail	San Francisco	Western



Sub-Query



Sub-Queries

- A select statement, used as an expression in part of another select, update, insert or delete
- The sub-query is resolved first and the results are substituted into the outer query's where clause
  - Used because it can be quicker to understand than a join
  - Perform otherwise impossible tasks (i.e. using an aggregate)

Example: Sub-Query

```
select product_num,
       description,
       purchase_cost
from product_tbl
group by product_num,
       description,
       purchase_cost
having purchase_cost >
  (select avg
    (shipping_cost)
     from order_tbl);
```

PRODUCT_NUM	DESCRIPTION	PURCHASE_COST
958888	486 133Mhz Computer	799.99
958889	486 90Mhz Computer	595.95
980001	Tax Application	1095
980005	Accounting Application	11500.99
980025	266Mhz Pentium Computer	2095.99
980031	RAD Kit for Windows 95	595.95
980122	133Mhz Pentium Computer	1400.95
980601	300Mhz Pentium Computer	2000.95
985510	21 inch Digital Monitor	595

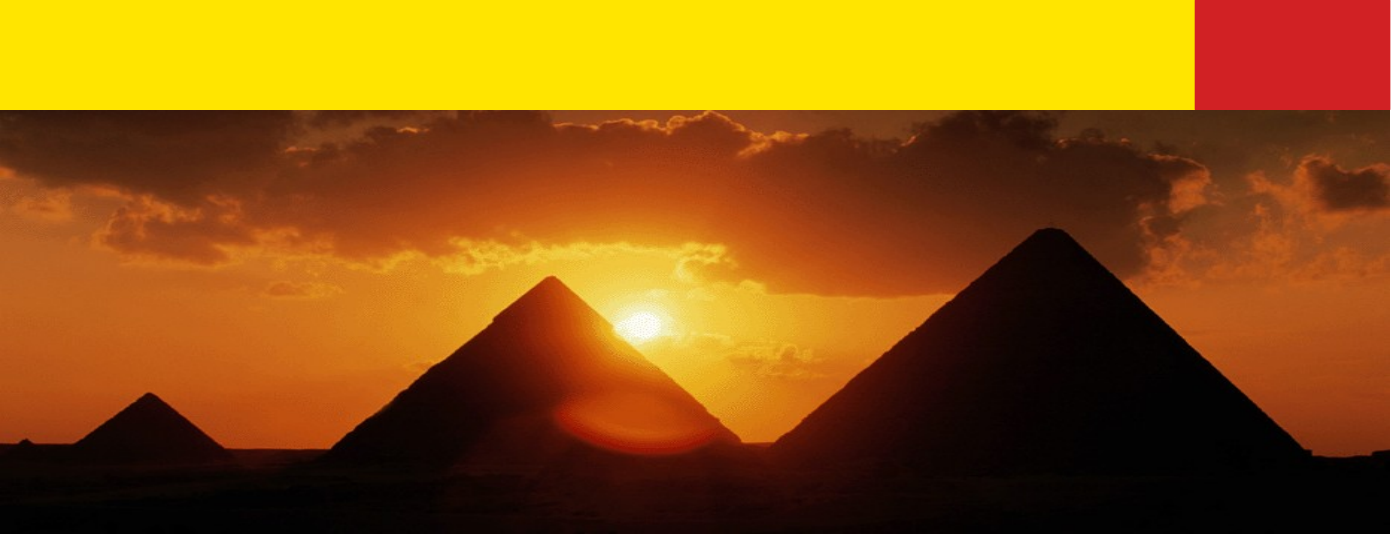
EXISTS and NOT EXISTS

- EXISTS and NOT EXISTS are for use only with sub-queries.
- They produce a simple true/false result
- EXISTS is true if and only if there exists at least one row in result table returned by sub-query
- It is false if sub-query returns an empty result table
- NOT EXISTS is the opposite of EXISTS

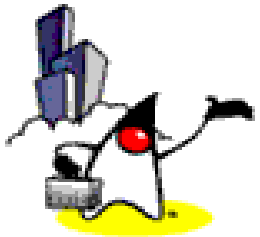
EXISTS cont.

```
select o.city,
        s.first_name || ' '||
    s.last_name
from sales_rep_tbl s,
        office_tbl o
where s.office_num =
    o.office_num
and exists
(select 1
   from manufacture_tbl m
where m.city = o.city);
```

CITY	s.first_name || ' '|| s.last_name
San Mateo	Heather Smith
San Francisco	George Valentine
Oakland	Jack Smith
New York	Frank Donohue
New York	Bill Show



Union, Intersection



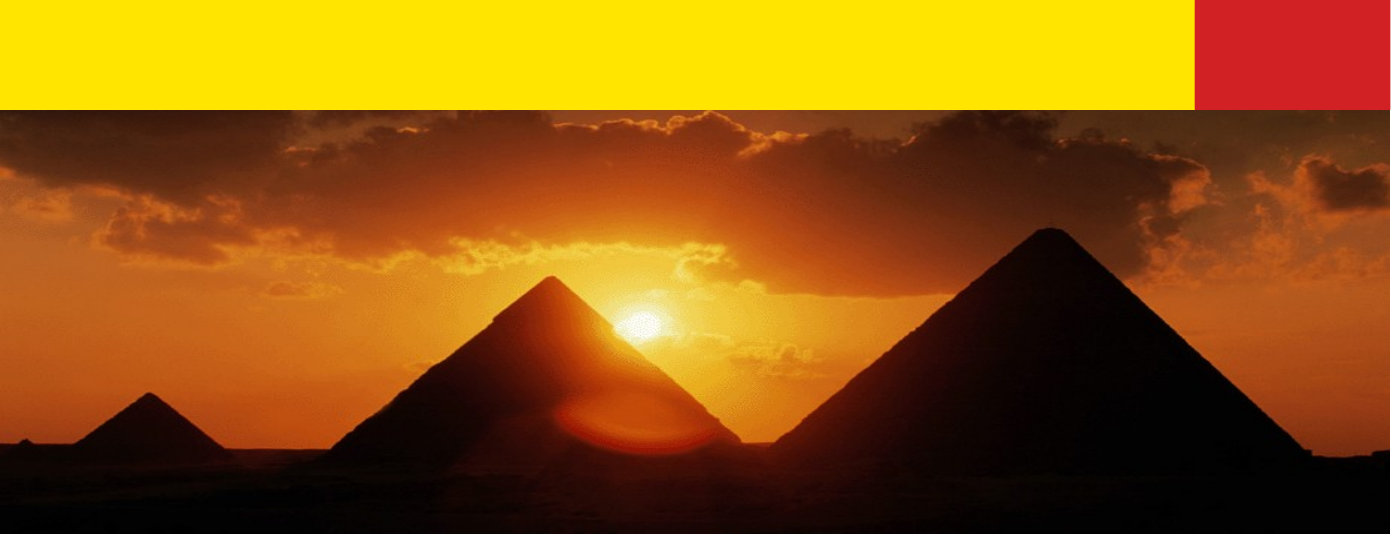
Union, Intersection and Difference

- Can use normal set operations of union, intersection, and difference to combine results of two or more queries into a single result table.
- Union of two tables, A and B, is table containing all rows in either A or B or both.
- Intersection is table containing all rows common to both A and B.

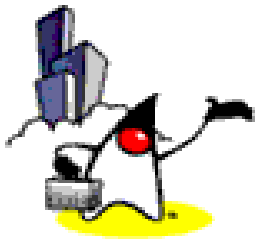
Union cont.

- Difference is table containing all rows in A but not in B.
- Two tables must be union compatible.

```
(SELECT area
FROM branch
WHERE area IS NOT NULL) UNION
(SELECT area
FROM property_for_rent
WHERE area IS NOT NULL);
```



Comparison, Character match, Numerical Expression



Comparison Operators

= equal to

> greater than

< less than

!= not equal

<> not equal

!> not greater than

!< not less than

>= greater than / equal to

<= less than / equal to

Character Matches

- Use to select rows containing field that match specified portions of character string
- Use with character data only
- Can use wildcards

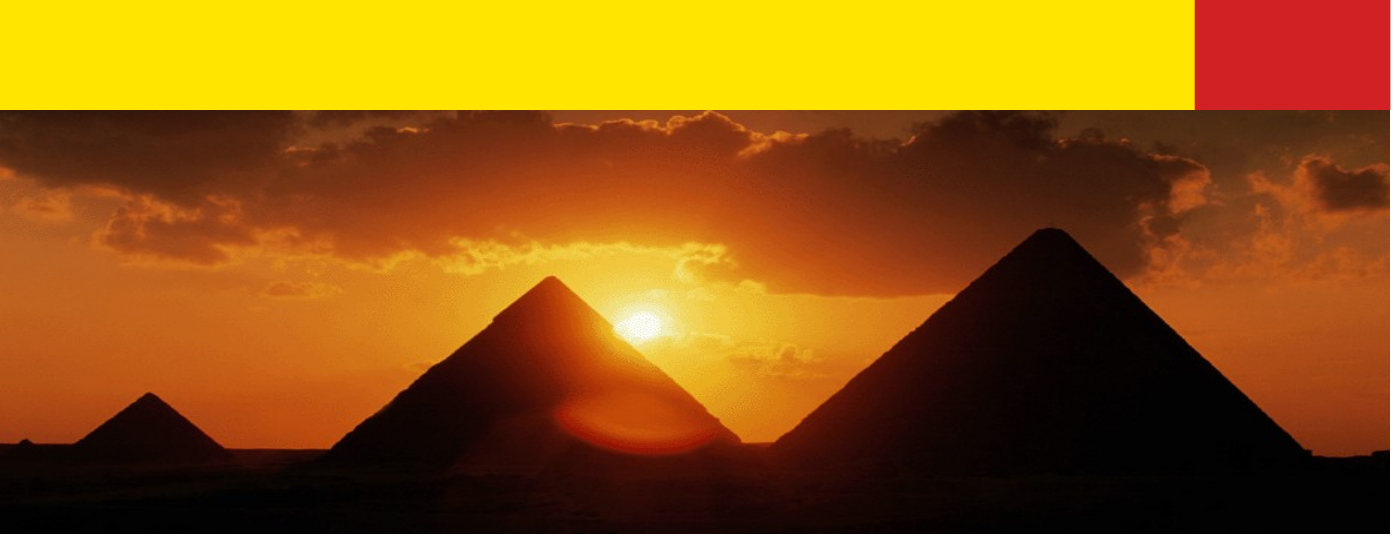
%	any string of zero or more characters

- Enclose wildcards and character strings in quotes

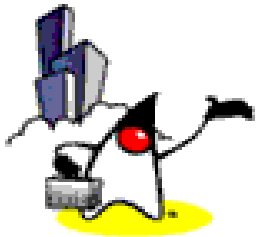
Numerical Expressions

- Numerical expressions can be used in any numeric column or in any clause that allows an expression

+	addition
-	subtration
*	multiplication
/	division
%	modulos



Aggregate Functions



Aggregate Functions

count(*)	number of selected rows
count(col_name)	number of nul-null values in column
max(col_name)	highest value in column
min(col_name)	lowest value in column
sum(col_name)	total of values in column
avg(col_name)	average of values in column

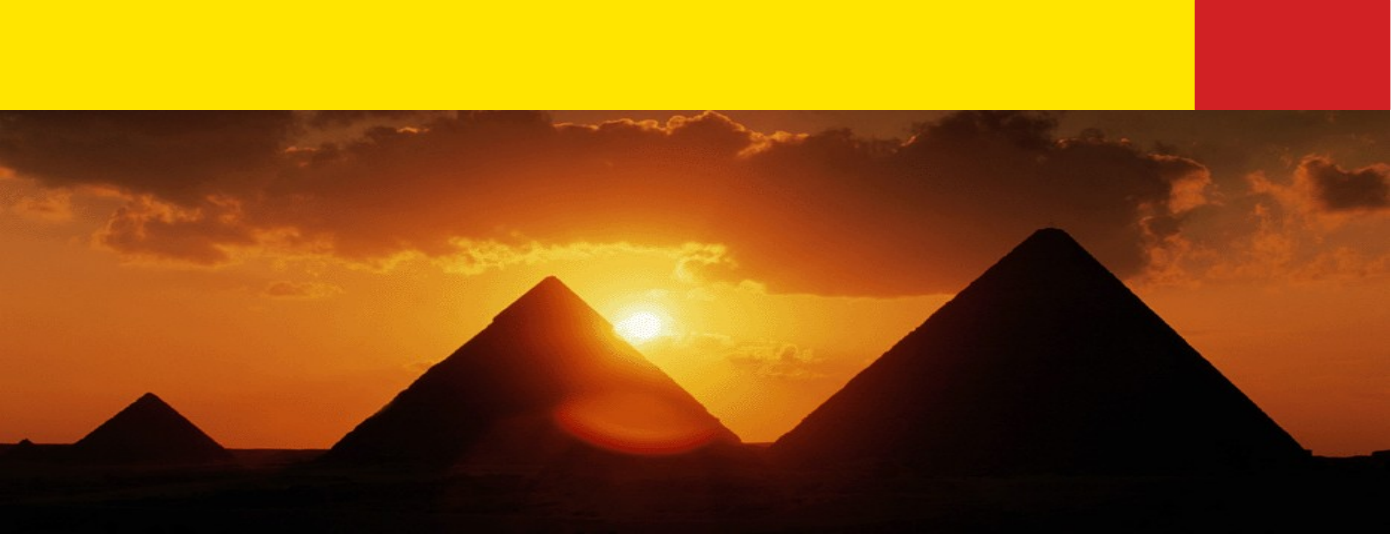
- Aggregates ignore null values (except count(*))
- **sum** and **avg** only work with numeric values

Aggregate Functions Cont.

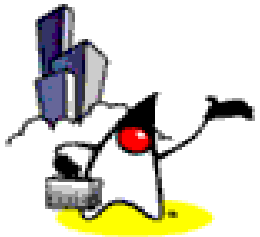
- If a **group by** clause is not used, only one row is returned
- Aggregates may not be used in a **where** clause
- Aggregates can be applied to all rows in a table or to a subset of a table
- Distinct keyword eliminates duplicate values before applying the aggregate function

NULL Values

- A **null** value is an unknown value
  - **null** is a special value which means “no information available”
  - **is null** should be used to match columns contains **null** values
  - “= **null**” can also be used, but is not recommended
  - One **null** value is never equal to another **null** value
  - **null**'s sort and group together
  - Some columns are defined to permit null values



Insert, Update, Delete



INSERT

```
INSERT INTO table_name [ ( col_name1, col_name2, .... ) ]
  VALUES ( expression1_1, expression1_2, .... ),
         ( expression2_1, expression2_2, .... ), ....

INSERT INTO table_name [ ( col_name1, col_name2, .... ) ]
  SELECT ...

INSERT INTO table_name
  SET col_name1 = expression1, col_name2 = expression2, ....
```

INSERT cont.

- The column list is optional.
- If omitted, SQL assumes a list of all columns in their original CREATE TABLE order.
- Any columns omitted must have been declared as NULL when table was created, unless DEFAULT was specified when creating column.

INSERT cont.

- Data value list must match column list as follows:
  - Number of items in each list must be the same.
  - Must be direct correspondence in position of items in two lists.
  - Data type of each item in data_value_list must be compatible with data type of corresponding column.

INSERT Examples

Not specifying column names:

INSERT INTO Dept VALUES (1, 'Engineering', ProjectSet());

Specifying column names:

INSERT INTO staff (sno, fname, lname, position, salary, bno)
VALUES ('SG44', 'Anne', 'Jones', 'Assistant', 8100, 'B3');

Insert using a select statement

INSERT INTO stores (store_name, total_sales)
SELECT ...

UPDATE

```
UPDATE table_name
  SET col_name1 = expression1, col_name2 = expression2, ....
  [ WHERE expression ]
  [ LIMIT limit_amount ]
```

- SET clause specifies names of one or more columns that are to be updated.
- **Where** clause is optional.
  - If omitted, named columns are updated for all rows in table.

Update Example

```
UPDATE emp
    SET sal = sal * 1.1
    WHERE empno NOT IN (SELECT empno FROM bonus);
```

```
UPDATE staff
SET salary = salary*1.05
WHERE position = 'Manager';
```

DELETE

DELETE FROM table_name

[WHERE expression]

[LIMIT limit_amount]

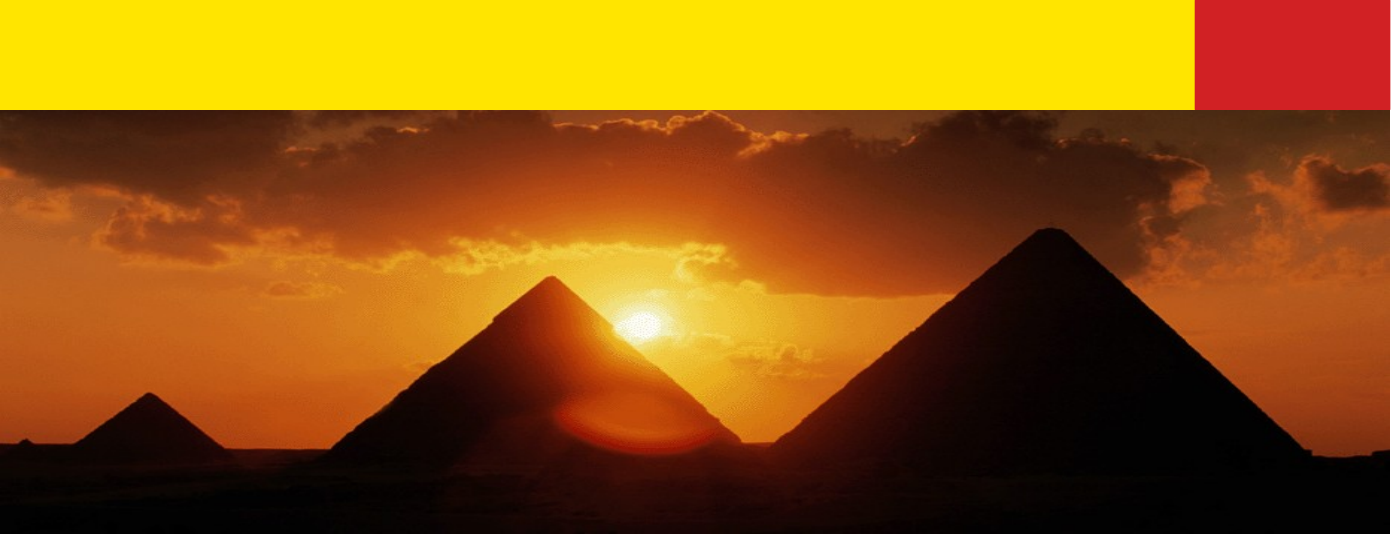
- **where** is optional; if omitted, all rows are deleted from table. This does not delete table. If the **where** is specified, only those rows that satisfy condition are deleted.

DELETE examples

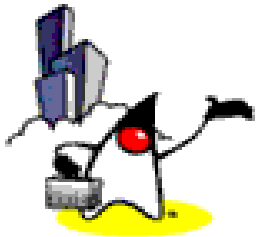
delete from store where store_num = 221;

Delete all information from the store table:

delete from store;

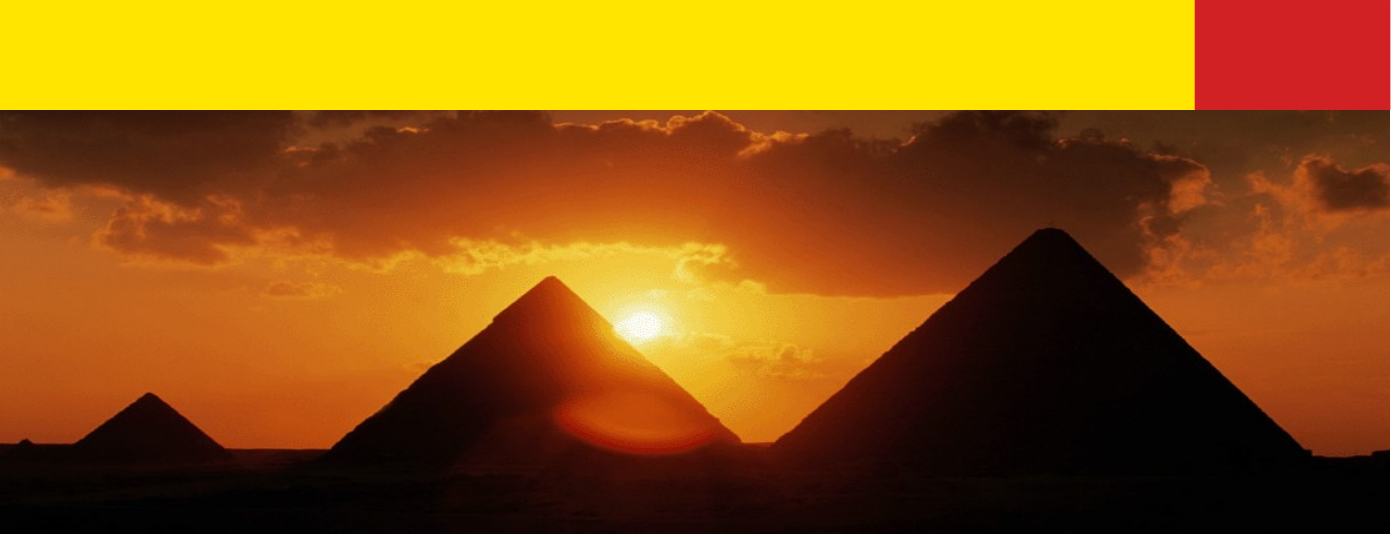


SQL Data Types



SQL Data Types

Data Type	Declaration
character	CHAR, VARCHAR
bit	BIT, BIT VARYING
numeric	NUMERIC, DECIMAL, INTEGER
approx. numeric	FLOAT, REAL, DOUBLE
datetime	DATE, TIME, TIMESTAMP



Passion!