Handout #7
Single Table Queries

SQL

SQL, originally called SEQuel, for Structured-English Query Language. A de-facto standard of relational model (although some people argue not good enough).

It’s both a Data Definition Language (DDL) and a Data Manipulation Language (DML).

Single table queries

In general, columns and various expressions and aggregate functions computed from the column values can be displayed.

Summary of the Syntax:

The general syntax is roughly as follows, but we will examine each part step by step.

```sql
SELECT column and/or computed column list, aggregate function list
FROM table list
WHERE condition
GROUP BY grouping attribute(s)
HAVING group condition
ORDER BY attribute list or group-aggregate function
;
```

Note the following:

1. all but the SELECT and the FROM clauses are optional, and
2. the order of evaluation is not top to bottom. See the summary in the end of this handout.

A few keywords

- **DISTINCT** operator - eliminates duplicates from the selected values,
- * denotes all,
- **NULL** denotes no value.

Now, further details of each part of the SELECT statement.
FROM

FROM is used to specify the table (or tables) from which the records (rows) are selected. We will consider single table queries first.

When the WHERE clause is missing - all rows are selected from the table.

FROM clause is always evaluated first.

SELECT in more detail

SELECT is used to narrow down the selection to certain columns, and computed values.

SELECT clause is always evaluated last.

Selecting columns:

The shortest query: selects all columns (*) (and all rows) from table htopi.customer_s:

```
SELECT *
FROM htopi.customer_s
```

Display only selected column values for each row: e.g. city and state from each row of the htopi.customer_s table:

```
SELECT cust_city, cust_state
FROM htopi.customer_s
```

Computed columns - columns that does not exist in the DB but can be computed using values from existing columns and mathematical operators +, -, /, *, e.g. The || operator concatenates string values.

```
SELECT cust_name, cust_city || ',' || cust_state, cust_phone
FROM htopi.customer_s
;

SELECT prod_name, prod_price * prod_qty_on_hand
FROM htopi.product_s
;
```

Aliasing: Column headings can be set as well, e.g.

```
SELECT prod_name AS Product, prod_price * prod_qty_on_hand AS TotalPrice
FROM htopi.product_s
```
Aggregate (Group) Functions SUM, AVG, MAX, MIN, COUNT.

Aggregate functions are applied to the values in a set of rows.

A single value is computed and displayed for each aggregate function, if no GROUP BY clause is present.

If aggregate functions are listed in the SELECT clause, the clause cannot include any columns, except those mentioned in GROUP BY. We will return to this when considering the GROUP BY clause.

Example: compute the total quantity on hand and average price of all products

```
SELECT SUM(prod_qty_on_hand), AVG(prod_price)
FROM htopi.product_s;
```

SUM returns the total sum of values in the specified numeric column from selected rows. Syntax:

- SUM (numeric column)
- SUM (DISTINCT numeric column)

NULL values are ignored.

AVG like SUM, but returns the average of values in the specified numeric column from selected rows.

MAX/MIN returns the greatest/lowest of the values in the specified column from selected rows. The column parameter is not limited to numeric columns.

Example: find the date of the most recent contact with a customer

```
SELECT MAX(cust_contact_date)
FROM htopi.customer_s;
```

COUNT returns a number of rows that contain a value (i.e. are not NULL) in a specified column. Has a special form that counts all rows, including those that have NULLS.

Syntax:

- COUNT (*) the special form that counts all rows
- COUNT (column). Ignores rows with NULL values in the column.
- COUNT (DISTINCT column). Ignores NULLS.

Examples: Count number of customer representatives currently serving customers (note the use of DISTINCT),

```
SELECT COUNT (DISTINCT cust_rep)
FROM htopi.customer_s;
```
Count how many customers are currently listed in the htopi.customer.s,

```
SELECT COUNT(*)
FROM htopi.customer.s;
```

Aggregate functions can be used over the computed columns as well as the real ones.

**Practice problems:** Write a query that

1. displays the total amount that can be derived from selling all products available on hand for the listed price,
2. displays the average profit (difference between price and cost) for each product listed.

**WHERE**

WHERE clause is used to **restrict the set of selected rows** by specifying a condition that must be true for a row to be selected.

Always evaluated after the FROM clause.

**Syntax:**

```
WHERE condition,
```

where the **condition** is a boolean expression (i.e. evaluates into true or false) that must be true for the record to be included in the selected set.

The following operators can be used in the condition:

1. Comparison operators (\(=, \!=(or <>), <, >, <=, >=\))
2. Comparison to NULL: **IS NULL**
3. Logical operators **AND, OR, NOT**
4. Set operators: **IN, EXISTS**
5. Pattern matching: **LIKE**

**Comparison Operators and IS NULL** are not limited to comparison of numeric values:

Examples: display product number and name for those products that have a price that's greater than 1000,

```
SELECT prod_nbr, prod_name
FROM htopi.product_s
WHERE prod_price > 1000;
```

display the names and phone numbers of all customers who have not been assigned a customer representative,
SELECT cust_name, cust_phone
FROM htopi.customer_s
WHERE cust_rep IS NULL;

display the phone number of customer 'Ezcopr' (note case-sensitivity w. respect to strings).

SELECT cust_phone
FROM htopi.customer_s
WHERE cust_name = 'Ezcorp';

Logical Operators: AND, OR, NOT - combine the results of simple conditions into compound conditions. For expressions A and B

- A AND B is true if and only if both A and B are true,
- A OR B is true if and only if at least one of A,B is true,
- NOT A is a negation of A - NOT true is false, NOT false is true.

In the expressions, NOT is evaluated first, then AND, and then OR.

Examples: display the average price of all products in the product line number 11 that have a cost equal to or greater than 200, but not greater than 1000.

SELECT prod_nbr, prod_name
FROM htopi.product_s
WHERE prod_prodline = 11 AND prod_cost >= 200 AND prod_cost <= 1000;

this last query can also be written using a BETWEEN operator to specify cost between 200 and 1000:

SELECT prod_nbr, prod_name
FROM htopi.product_s
WHERE prod_prodline = 11 AND prod_cost BETWEEN 200 AND 1000;

display the names of all customers from California and Texas

SELECT cust_name
FROM htopi.customer_s
WHERE cust_state = 'CA' OR cust_state = 'TX'

Set membership: IN

Syntax: column-name IN (setOfValues).

Examples: the set of values can be given as a list, e.g.
SELECT prod_nbr, prod_name
FROM htopi.product
WHERE prod_prodlne IN (3,11,18);

SELECT cust_name
FROM htopi.customer
WHERE cust_state IN ('CA','TX')

or as a result of another query (subquery). The example below demonstrates that.

List the names and states of all customers who have a customer representative that serves a customer from Connecticut.

SELECT cust_name, cust_state FROM htopi.customer
WHERE cust_rep IN ( SELECT cust_rep
FROM htopi.customer
WHERE cust_state = 'CT' )

The inner query (subquery) is evaluated first. Its results are later used to filter out the rows of the outer (top) query.

**Pattern matching: LIKE**

**Syntax:** column LIKE (pattern-string), where pattern-string may use the following wildcard symbols:

- `%` represents any collection of characters,
- `_` represents any single character.

The value in the column is checked against the pattern and if the value matches the pattern, the row is selected, otherwise it is excluded.

**Examples:** list the average price of monitors listed in the htopi.product table:

```
SELECT AVG(prod_price)
FROM htopi.product
WHERE prod_name LIKE '%Monitor%' OR prod_name LIKE '%monitor%';
```

list the contact date and customer name of all customers whose last contact date was not in 2002,

```
SELECT cust_name, cust_contact_date
FROM htopi.customer
WHERE TO_CHAR(cust_contact_date, 'MM-DD-YY') NOT LIKE '%2002'
```

Note the use of the NOT operator. Note also how the date, that is by default represented in the format MM-DD-YY is converted using the TO_CHAR function into a string of format 'MM-DD-YYYY' to be used with the LIKE operator.
GROUP BY

GROUP BY clause is used to group rows to compute group-statistics. Evaluated after the WHERE clause.

Consider the following query:

Display the total price of products for every product line:

Must group products with the same product line together and then compute the total price for each group.

Syntax: GROUP BY grouping-column(s)

groups all rows that have the same value in each of the grouping-columns together. The aggregate function in the SELECT clause is then applied to members of each separate group.

Example: display the total price of products for every product line. Include the product line in each row:

```
SELECT prod.prodline, SUM(prod.price)
FROM htopi.product
GROUP BY prod.prodline
```

Note: when the GROUP BY clause is present the SELECT clause may include only the columns that appear in the GROUP BY clause and aggregate functions.

Here’s another example - can you deduce what’s returned?

```
SELECT cust.cust_group, count(*)
FROM htopi.customer
WHERE cust.state = 'CA'
GROUP BY cust.cust_group
```

HAVING

HAVING restricts the groups according to a specified condition.

Different from WHERE, HAVING can use aggregate functions (they are not allowed in the WHERE clause).

Syntax: HAVING group condition,

where the (group condition) is similar to the condition defined for the WHERE clause, but can also use aggregate functions that apply to members of each individual group.

Ex. How to formulate the following query?

For those product lines in which the average price of the product is under 500, display the product line number and total number of product on hand.

Must
1. group products by product line, (GROUP BY )

2. compute the average price in each group, and consider only those groups that have average price under 500 (HAVING ),

3. display the product line number and total number of products on hand (SELECT ).

Example: for those product lines in which the average price of the product is under 500, display the product line number and total number of product on hand

```sql
SELECT prod.prodline, SUM(prod.qty_on_hand)
FROM htopi.product
GROUP BY prod.prodline
HAVING AVG(prod.price) < 500
```

The HAVING clause for groups serves the same role as the WHERE clause for individual rows: it selects only those groups that satisfy a certain group condition. Since the WHERE cannot use aggregate functions, it cannot be used to filter out groups in the way HAVING clause does.

**ORDER BY**

sorts the rows in increasing (default) or decreasing (DESC) order of the values in specified columns. Evaluated after HAVING .

Examples:

```sql
SELECT prod_nbr, prod_name
FROM htopi.product_s
WHERE prod.prodline = 11 AND prod_cost BETWEEN 200 AND 1000;
ORDER BY prod_nbr DESC
```

```sql
SELECT prod.prodline, AVG(prod.price)
FROM htopi.product_s
GROUP BY prod.prodline
ORDER BY AVG(prod.price)
```
Summary of the SELECT statement evaluation model

Recall the general form of a SELECT query:

```
SELECT column and/or computed column list, aggregate function list
FROM table list
WHERE condition
GROUP BY grouping attribute(s)
HAVING group condition
ORDER BY attribute list or group-aggregate function
```

Here's a summary describing how it is evaluated conceptually. The order of items is important.

- Apply the FROM clause to identify the tables involved, and to create any joined tables.
- For each row of the participating tables or joins evaluate the condition of the WHERE clause. Include only those rows for which the condition evaluates to TRUE.
- Apply the GROUP BY clause to group together those of the included rows that have identical values of the grouping attribute(s).
- Include only those groups that satisfy the group condition of the HAVING clause. The group (or aggregate) functions (SUM, AVG, COUNT, etc.) are applied to each group formed by the GROUP BY clause separately.
- Evaluate the ORDER BY clause by sorting the included rows in the specified order (ASC(default) or DESC) of the values in attribute list.
- For each of included rows, select only the columns and computed columns that appear in the SELECT clause.
  The group functions are applied to each group formed by the GROUP BY clause separately. When the GROUP BY clause is absent, they are applied to the entire set.